Department of Computer Science

Neurythmic: a musical creation tool based on central pattern generators

Daniel Thomas Bennett

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2017 | CSMSC-17

Declaration:

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Daniel Thomas Bennett, September 2017

Acknowledgements

I would like to thank my thesis supervisors, Anne Roudaut and Pete Bennett, for their help, good advice and support throughout the project. I learned a huge amount from both.

Thanks also to my colleagues at University Hospital Bristol who were so generous and supportive in making space for me to complete this degree.

Thanks to Robin Allender, Seth Cooke, Paul Jebanasam and Dominic Lash for their participation in testing, their experience and their intelligent, analytical feedback.

And most of all, thanks to Jess Clark for more or less everything else.

Contents

Chapter 1
Executive Summary

This thesis describes the development of Neurythmic: a system for creating music with Central Pattern Generators (CPGs).

Neurythmic is the first interactive music tool to use CPGs, and this thesis is the first example in the literature to focus on interactive musical rhythm creation with CPGs.

CPGs are neural networks which generate self-sustaining rhythms, which adapt to input. They are found in animals, where they generate phenomena such as heartbeat, gut-peristalsis and support motor control. In robotics they are used for a variety of purposes from gait generation to traffic light control.

This project uses these systems (in particular a variant called Matsuoka's Neural Oscillator) to support the interactive creation and control of fluid, unquantised musical rhythm. This approach can generate subtle, or dramatic variation over time and avoids the rigidity of timing which can be characteristic of conventional step-sequencer outputs. In user testing with musicians it was shown to be a versatile tool, open to a range of quite different musical approaches.

Neurythmic was developed by working closely with expert musicians. The techniques used in that creative collaboration - a combination of explicitation interviews, think aloud studies and thematic analysis - are also described.

The main contribution of this thesis is the Neurythmic application itself. Additional contributions include:

- A CPG library for the MAX/MSP music creation environment, enabling further musical experimentation by non-programmers.
- Models for musical application of CPG networks
- A method for accurately controlling frequency of Matsuoka's Neural Oscillator.
- A result for deriving the stopping threshold of Matsuoka's Neural Oscillator.

Chapter 2
Introduction

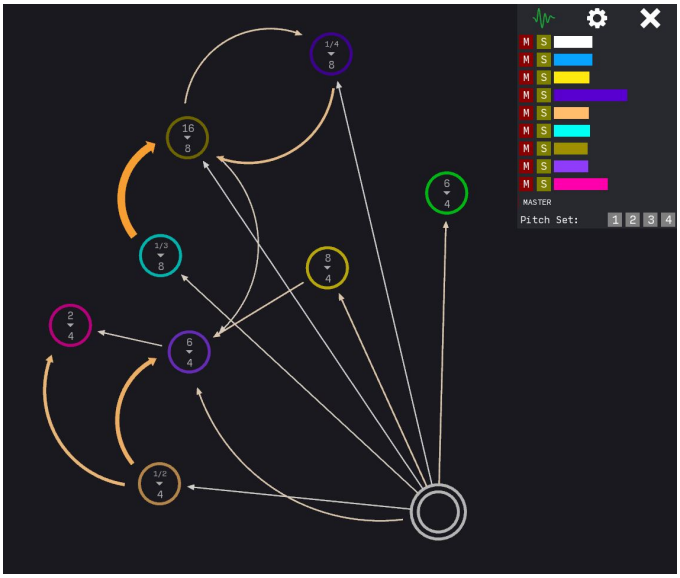

Figure 2.1: Screenshot of the final version of the Neurythmic system presented in this project

Background

Music making is as old, and as widespread, an impulse as almost anything in human nature; so the advent of the computer was inevitably followed by the advent of computer music. As computer technology progressed, so did the possibilities for computer music making, to the point that, now, these tools are behind almost all of our popular music, and much other music also. The forms of these tools can have a decisive impact on the kinds of music produced.

Recent years in particular have seen a surge in the popularity of improvisatory, real-time, creation of musical rhythms using electronic sequencers - music composition tools that allow users to specify sequences of musical events in time. Where at one stage the primary mode of computer music interaction was off-line and compositional. Subsequent advances in technology and interface design have allowed musicians to work with electronic music in ways that are interactive, improvisatory and performative; analogous to the ways in which we work with conventional instruments.

Music sequencers and drum machines are, of course, quite different from conventional instruments, in certain ways at least. The most striking difference is perhaps in their relationship to musical gesture. When playing a guitar or violin the player directly produces the sound - each physical action on the instrument corresponds quite directly to a musical event. With a music sequencer or drum-machine this relationship is broken. Here the

domain of control and the mode of performance is more detached and compositional, a matter of composing, moulding and altering processes that unfold in time.

There are advantages to this more detached approach, however. A single musician can create a piece of music for multiple voices by improvising and progressively refining an idea. With some refinement and practice, live performance with sequencer systems is also possible, and in fact increasingly popular (witness the growth in popularity of Algorave events and similar [1]). However there are also trade-offs - these come in the nuance of the musical results, in their reliance on particular identifiable forms of musical variation, on rigid repetition, and on rigid, quite inflexible timing.

Aesthetics being what they are, musical forms have developed around these qualities, and thriving musical cultures exist exploiting precisely the mechanical qualities of electronic sequencing interfaces. Nonetheless there is undoubtedly value in overcoming these limitations and offering new modes of expression. These need not supplant existing modes, but rather augment and enrich them.

Neurythmic

The current thesis describes the development of a new musical tool, Neurythmic, which responds to the situation described above. A screen-shot from the final version of this can be seen in fig. 2.1 above.

Neurythmic offers a new means of electronic rhythmic creation and performance; it is an instrument in the mould of sequencers and drum machines but quite different in form. It aims to support the intuitive, interactive creation and control of rhythm, in a way that avoids rigidity of timing, and encourages the creation of subtle rhythmic variation. In conventional music sequencing, musicians input notes directly, onto a coarse, time-quantised grid. In Neurythmic, by contrast the musician builds musical systems that generate patterns and which can be moulded to produce particular results and even performed.

Neurythmic generates rhythms based on the output of a Central Pattern Generator (CPG) network. CPGs are neural networks which generate adaptive rhythms. They are found in animals where they underly heartbeat, lung function and peristalsis, as well as complex motor control systems such as gait. Their generation of self-sustaining, multi-phase rhythms, combined with their ability to adapt these rhythms to sensory input has led to their wide adoption in robotics. Here they are used for everything from gait coordination to the control of traffic lights.

Neurythmic is the first music tool in the literature to use the adaptive properties of CPG networks for interactive musical rhythm generation. The goal of using CPGs here is threefold:

- To support real-time, improvisatory control of musical rhythm which does not rely on underlying time grids and so allows for more fluid specification of time.
- To support generation of subtle (or even dramatic) rhythmic variation over a number of bars
- To offer a new means of interaction with musical rhythm, which may in turn help musicians to find new approaches to their work

Only one previous paper discusses the use of CPGs for music [2]. That paper focused on the direct generation of musical timbres, and limited its analysis to feed-forward relationships in pairs of half-centre oscillators. By contrast, this thesis focuses on rhythm creation, works with larger networks and begins to explore interaction patterns, network structures and behaviours which are of particular relevance for creative musical applications. As a

contribution in a new area of research, this thesis aims to support further work in this area by documenting network behaviour and structure, and by documenting musical models of use which may support further research.

Collaborative Design

In developing a novel musical instrument in an area with little precedent, there is a risk that the output becomes overly idiosyncratic - tied to the tastes, practices and imaginative limitations of the researcher. I have tackled that problem by using a user-centred design approach - a method commonly used in the design of interactive systems. This incorporates feedback from others at every stage of the project, beginning with informal testing with friends during prototyping, and progressing to a main development process, structured around formal testing.

I recruited four expert musicians - three professionals and one expert-amateur - to provide feedback and expertise. I worked with them using observed creative tasks, explicitation interviews and think-aloud usability studies focusing on Neurythmic. This process was integral to the success of the project, and is described in more detail in chapter 9.

Project Map

The project divides into four parts -

(I) **Literature Review:** Here I discuss the context of Neurythmic and results from previous work which were useful in its development.
In ch.3 I discuss relevant literature on CPGs, specifically focusing on the basic CPG unit used in this project - the two neuron, half-centre version of Matsuoka's Neural Oscillator.
In ch.4 I discuss previous approaches to real-time rhythm sequencing, placing their outputs in a taxonomy of rhythmic spaces drawn from [3].
In ch.5 I discuss literature on the design of digital musical instruments, covering development approaches and design principles.

(II) **Analysis & Prototyping:** In this section I discuss the groundwork for development.
In ch.6 I discuss the prototyping phase of the development of Neurythmic. I discuss the development of the Matsuoka's Neural Oscillator library for Max/MSP which supported this, and the testing rig built around it. I discuss the process of idea generation and testing, and outline those results from this phase which fed into development.
In ch.7 I discuss the simulation and analysis of networks of Matsuoka's Neural Oscillator (MNO) - the type of CPG used in this project. I describe a method for accurately controlling the frequency of the oscillator.

(III) **Development**: In this section I discuss the process of developing the Neurythmic app in dialogue with 4 expert musicians, in a process which divides into two phases: First I develop an initial version of the application and subject it to formal formative testing. Then I develop the final Neurythmic system based on the results of formative testing and network analysis.
In ch.8 I describe the tools used in this project.
In ch.9 I describe the methods used to structure dialogue and collaboration with four musicians as part of a user-centred design process. I introduce the musicians, and

describe protocols for a think aloud study and for an expert-user task followed by explicitation interview.
In ch.10 I describe the process of designing and developing the Neurythmic application, and outline its pertinent features.

(IV) **Outcomes:** In this section I describe the formal summative analysis of the final Neurythmic system. I describe useful models for working creatively with CPG networks, and finally discuss directions for future work.
In ch.11 I describe the protocol used for final evaluation, (mirroring that of the expert user task in ch.9), and present the results. Results are arrived at by analysis of interview transcripts from the study using the thematic analysis technique.
in ch.12 I describe models I found useful for creative practice with CPG networks
in ch.13 I discuss future directions for research in this area

Project Outputs

The main contribution of this thesis is the Neurythmic application itself. Additional contributions include:

- A CPG library for the MAX/MSP music creation environment, enabling further musical experimentation by non-programmers. (ch.6)
- Models for musical application of CPG networks (ch.6)
- A method for accurately controlling frequency of Matsuoka's Neural Oscillator CPGs. (section 6)
- A result for deriving the stopping threshold of Matsuoka's Neural Oscillator, and description of behaviour on stopping and restarting (section 7)

Part I

Literature Review

Chapter 3

Central Pattern Generators and Matsuoka's Neural Oscillator

Overview

Central Pattern Generators (CPGs) are neural networks - biological or synthetic - that generate self sustaining but adaptive, rhythmically patterned outputs. In animals, CPGs underly many rhythmically coordinated behaviours, including breathing, chewing and gait[4]. A classic example of study in this area is the stomatogastric ganglion of crabs, thanks to their capacity to sustain activity *in vitro* [5]. In robotics CPGs are used in a wide range of motor-control applications.

The literature on synthetic CPGs (hereafter I will use "CPGs" to refer to the synthetic, computational variety) is extensive, covering applications from robotic arm control [6] to the efficient coordination of traffic signals [7]. Furthermore there are many models of CPG, modelling neurons in greater or lesser detail, and describing various conditions and topologies[4]. A common model, found in many variants of CPG is known as "half centre". This describes two populations of neurons, coupled in a mutually inhibitory relationship. A model of this kind, specifically a variant of Matsuoka's Neural Oscillator[8], will be explored in this project, and is the focus of the remainder of this chapter.

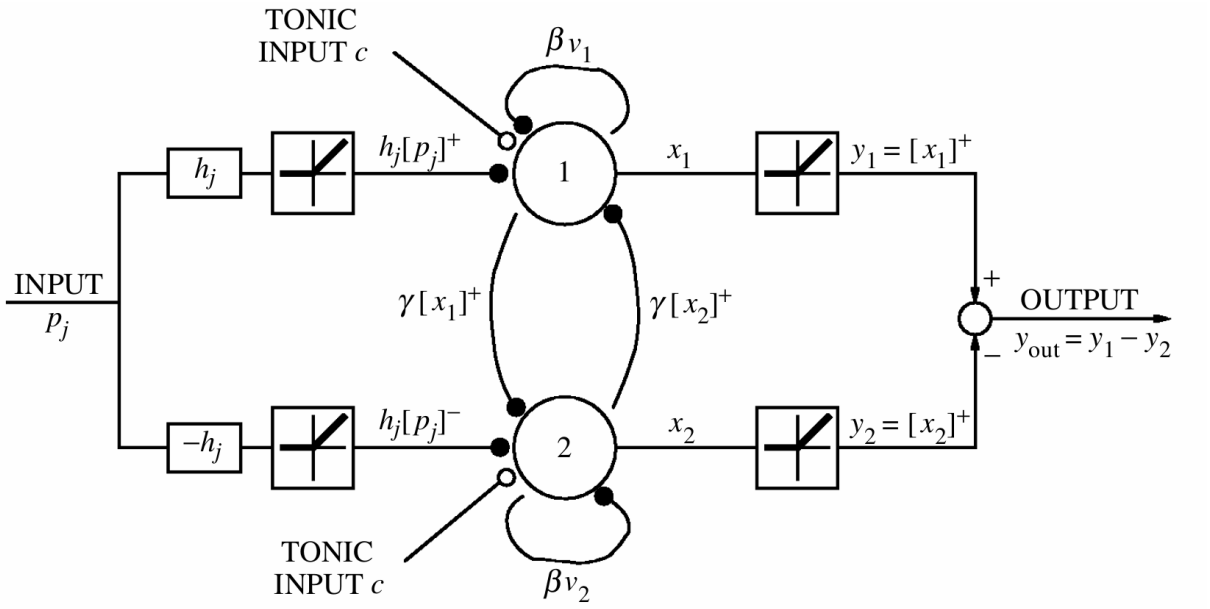

Figure 3.1: Two neuron, half-centre Matsuoka Oscillator - this model, with slight variations, used in all examples discussed (diagram taken from [6])

Matsuoka's Neural Oscillator

The only previous paper focusing on generation of musical material with CPGs, uses a variety called Matsuoka's Neural Oscillator(MNO)3. To build on this research and on the extensive, relevant literature on non-musical applications of the same oscillator, I use MNO in this project also.

There are various forms of MNO, taking in different numbers of neurons and different topologies[8][9]. Each form is described by a set of non-linear differential equations, which models the behaviour of a set of neurons connected by inhibitory synapses and capable, as a group, of giving rise to oscillatory behaviour. The MNO variant which recurs most often in the literature, and which is used in my own project, is the simplest - a variant, modelling just two neurons in a relationship of mutual inhibition (fig.3.1). This model takes one or more excitatory inputs and a summed inhibitory input. The equations for this variant are detailed in fig.3.2.

This basic two neuron structure is able to entrain to periodic input signals - often from another MNO unit. By "entrainment" here we mean that a system can bring its own oscillation into synchronisation with that of an input signal.

Since this entrainment will naturally affect the frequency of the oscillator output, it is useful to be able to speak of the effective output frequency due to entrainment, separately from the frequency of the node as it would have been without entrainment. As such, in the rest of this project I use the term **"natural frequency"** to refer to the frequency of the oscillator at a particular parameter state, in the absence of input.

$$\tau_1 \dot{x}_1 = c - x_1 - \beta v_1 - \gamma [x_2]^+ - \sum_j h_j [p_j]^+, \tag{3.1}$$

$$\tau_2 \dot{v}_1 = [x_1]^+ - v_1, \tag{3.2}$$

$$\tau_1 \dot{x}_2 = c - x_2 - \beta v_2 - \gamma [x_1]^+ - \sum_j h_j [p_j]^-, \tag{3.3}$$

$$\tau_2 \dot{v}_2 = [x_2]^+ - v_2, \tag{3.4}$$

$$y_{out} = [x_1]^+ - [x_2]^+. \tag{3.5}$$

Figure 3.2: Equations for the half-centre Matsuoka Oscillator

Description of the Equations

In the equations presented in (fig.3.2.) $[x]^+$ stands for the function $max(x, 0)$, and $[x]^-$ for the function $max(-x, 0)$. Subscript numbers indicate the neurons to which parameters belong. x is the membrane potential of the neuron, v represents the state of adaptation of the neuron. τ is the time constant, determining the reaction time. β is the constant determining adaptation time of both neurons and γ the constant determining degree of adaptation of the neurons to one-another. c is the "tonic", excitatory input, determining the amplitude of oscillation. p is the set of j inhibitory external inputs to the neuron, its positive part affecting neuron 1, its negative part neuron 2 h denotes the weights of each of these inputs. Finally, y_{out} is the output of the system

CPGs for Music

Excepting a brief paragraph in [10], the only research to date on the generation of musical material with CPGs seems to be that of Alice Eldridge. Her paper on the subject, *Neural Oscillator Synthesis*[2] (which appears to be unpublished, though it is cited several times in published papers (e.g. [10][11])) focuses on the use of the two neuron MNO described above, primarily at audio frequencies, for the synthesis of complex timbres. While this project uses MNO for the generation of rhythm, many of Eldridge's findings remain applicable.

Eldridge's analysis of the MNO focuses mainly on curves describing entrainment behaviour in a 2 oscillator feed-forward relationship, and on frequency spectrum of output when the oscillator is used without input, while varying those equation parameters which control wave-shape. This thesis takes a step beyond this by analysing larger networks, and building systems for interaction with them.

Wave-shape

Eldridge shows that the wave-shape of an MNO output can be varied from something close to a sine, to something close to a pulse (amplitude of harmonics decreasing in inverse proportion to their ordinal numbers). Control of wave-shape is useful when networking MNO instances. I find in my own analysis (see 7) that shape of input signal has a significant effect on entrainment behaviour.

Eldridge notes that it is difficult to separate control of wave-shape and frequency. Later in this chapter we will, however, see results which support this separation, and in 6 I will describe a method I developed for quite accurate control of frequency.

Entrainment Threshold

Eldridge's paper also presents graphs showing that the threshold for entrainment behaviour is based on the ratio of input signal to an oscillator's natural frequency. This threshold is lowest where the node's natural frequency is close to that of the incoming signal, and shows another minimum at around half this, with steep rises either side of these minimums.

In ch.7 I generate such curves for myself, specific to the tuning of my own system, to support user control interfaces.

Phase Lag at Entrainment

Eldridge also describes the amount of phase lag between input signal and entrained node for the tuning simulated. She does not discuss the effect on phase lag of frequency, wave-shape of input, nor the consistency of final entrained phase relationship across starting phase relationships. Since beat placement in my system will depend on such phase relationships, I address these questions in 7.

Sub-entrainment Behaviour

At input weights below entrainment, Eldridge notes there is still some entrainment effect, though she describes this as "unstable" or "turbulent", having the character of a system jumping between two or more attractors. She notes that at "natural frequencies" in the audio spectrum this is heard as a "rough acoustic texture".

In the roughly 0.1-5hz range used in this project, this same "attractor-jumping" effect is heard as an interesting, semi-predictable, rhythmic variation. It is one of the properties which makes me find these systems appealing for rhythm generation. Further analysis of behaviour in this range is found in ch.7.

Topology and Stability

The most useful resource I have found on the topology and stability of MNO is Matsuoka's own 1985 paper on the subject[8]. Though this paper is 30 years old at time of writing, Matsuoka himself refers back to its results in a 2011 paper discussed in the last part of this section [12] and remarks on the lack of other literature analysing the tuning of the oscillator.

In this paper, Matsuoka describes the differing properties of 11 variants of MNO, from the 2 neuron model under discussion in this chapter through various topologies possible with 3 to 6 neurons. He notes that within these topologies there are stable and unstable types - those that continue oscillating even under large disturbances, and those that may not. Matsuoka notes that it is a condition of stability that, within a network, there be a subgroup, D, of neurons which are not connected to one another, and that every neuron not in D receives at least one inhibitory signal from a member of D.

In my own system I use the 2 neuron variant pictured on p.7, connecting these units one to another via the summed output. This unit, containing no D subgroup, is not itself stable on the definition given above, but Matsuoka defines conditions for its stability[8]. For the 2 neuron MNO variant as discussed in this project the stability conditions can be expressed as in fig.3.3.

$$\frac{\gamma}{1+\beta} < c, \qquad \gamma > 1 + \frac{\tau_1}{\tau_2}$$

Figure 3.3: Stability conditions for two neuron Matsuoka Oscillator[8]

In this project I use the oscillator within these bounds of stability, building constraints into my implementation based on this result.

Tuning the Matsuoka Oscillator parameters

Prediction of Frequency and Amplitude

In his 2011 paper on analysis of MNO [12] Matsuoka provides an equation to estimate the angular frequency of the oscillator, demonstrated by the author as accurate for values of γ up to 1.0 and for values of β up to about 3.0. Where β and γ are equal this equation simplifies to the inverse square root of the product of the two time constants

$$\omega_n \approx \frac{1}{\tau_2}\sqrt{\frac{(\tau_1+\tau_2)\beta-\tau_1\gamma}{\tau_1\gamma}} \qquad \text{or} \qquad \omega_n \approx \frac{1}{\sqrt{\tau_1\tau_2}} \text{ (where } \gamma=\beta)$$

Figure 3.4: Approximation of Frequency for 2 neuron Matsuoka Oscillator[12]

Later, in 6, I describe developing a simple procedure to improve the accuracy of this result so that it can be used for direct control of frequency in my system.

A second result in [12] gives the amplitude of the oscillator, though the author indicates that this deviates further from simulated results than the equation for frequency - giving a reasonable estimation for values of τ_2 below about 0.3, and γ below about 2.4 .

$$A_x \approx \frac{c}{2\frac{\tau_1+\tau_2}{\tau_2\gamma} - 1 + \frac{2}{\pi}(\gamma+\beta)\sin^{-1}(\frac{\tau_1+\tau_2}{\tau_2\gamma})}$$

Figure 3.5: Approximation of Amplitude for 2 neuron Matsuoka Oscillator[12]

This result indicates that amplitude increases with τ_1 and γ, and decreases with τ_2 and β. Since, as discussed above, entrainment of the oscillator to input is dependent on input amplitude, this result could be used, for example, to normalise output amplitudes of all nodes to ensure predictable entrainment independent of parameters. This was not found necessary in the current project.

Tuning by "Feel"

It is a condition of my project's success that, with a suitable control mechanism, users should be able to control networks of MNO intuitively, achieving attractive musical results.

In the paper discussed in the previous section [12] Matsuoka noted that the literature was lacking concrete guidelines other than his own for even very basic questions around tuning the oscillator. Given the extensive and quite successful application of MNO, Matsuoka notes that people must be applying the oscillator in the absence of basic guidelines. This suggests some kind of manual tuning may be common.

More evidence for this is found in Libera et al[13]. They present a system which allows the intuitive tuning of CPGs for gait. Here a small humanoid walking robot is designed such that users can tune gait by providing touch feedback on the robot's limbs - guiding the model towards the desired gait pattern.

The authors review literature on tuning CPGs using genetic algorithms and learning approaches, but argue that in many real world cases, the goals for CPG tuning are not easily specifiable in terms of fitness functions. They give as an example the goal of "humanness" of gait. Here they suggest an outcome is most immediately understood in terms of its "feel", and this makes it easier to progressively guide a system towards a desired result than to specify one in advance. This has obvious resonance for Neurythmic and other creative applications of CPGs.

The connection in approach here leads me to propose a future area of research leading from my own project. Insofar as Neurythmic offers a model for intuitive specification of CPG behaviour, it may prove adaptable as an approach for CPG tuning in robotics.

Chapter 4
Computer Music Context

In this chapter I discuss the computer-music context for Neurhythmic, focusing on other approaches to rhythm sequencing in the same area. I discuss both general approaches and specific cases, covering precedents which give context to the current research, and examples which were instructive in my own design.

Before discussing these examples I will briefly outline the wider musical context: specifically I introduce a taxonomy of "rhythmic spaces" which will be useful for clarifying distinctions between the systems discussed

Rhythmic Spaces

In discussing various examples from literature below, and relating their merits and demerits, I will use of a taxonomy "rhythmic spaces" developed by composer and programmer Curtis Roads. This appears in his recent book *Composing electronic music* [14], and the taxonomy could be fairly described as computer-music-centric account. Roads outlines the approaches to rhythm as follows:

1. Aligned to a constant metrical grid
2. Aligned to more than one simultaneous metric grid
3. Aligned to a variable metrical grid
4. Aligned to more than one variable metrical grid
5. Gravitating around points of attraction (y) and away from points of repulsion (x)
6. Unaligned to a metrical grid or gravitational points
7. Combinations of (1) to (6)

The first case here describes the primary domain of the majority of approaches to improvisatory or performance-centric rhythm sequencing. Much mainstream electronic pop music fits into this category too, arguably as a result.

By contrast 20th century art music, and increasingly experimental pop music give plentiful examples of music, moving in and between each of the other spaces, from Autechre, through Bjork and Bernard Parmegiani to the "microsound" compositions of Curtis Roads himself[15]. Neurhythmic aims to provide a system adequate to all of these spaces, and to movement between them.

Performance Music Sequencers

If musical performance and improvisation are ancient, then their pursuit using music sequencers is relatively new - even in the context of the relatively young discipline of electronic music.

Early in the development of computer music, performance with the machines was not possible due to lack of computing power. Curtis Roads notes that composers were nonetheless drawn to the computer, and often by the possibilities it offered for precise specification of musical time. Ironically, this very precision, in the absence of suitable interfaces proved a barrier to "naturally flowing rhythm" and "immediacy of expression" [14].

The "Classic" Step Sequencer

As interfaces for interactive rhythm specification began to be developed, they addressed the issue of "immediacy of expression" by the common interface-optimisation of applying constraints. This optimisation came at the cost of "naturally flowing rhythm".

Following the pattern of musical notation, nearly all interactive rhythm sequencing interfaces use a metric grid representation of time. In most such interfaces designed for performance and hands-on, realtime creation, this metric grid generally represents a single, bar - looping or sequenced alongside other bars - divided by a fixed number of discrete time steps - generally 16.

While optimising for ease of fast rhythm specification, this approach places considerable limitations on the range and character of rhythms that can be created. While obviously restricted to the first of Roads' rhythmic spaces, their inflexibility goes beyond this. A 16 step bar divides neatly into 2, 4 or 8 parts, and so "common time" rhythms are easily specified. 16 does not, however, divide by 3, 5, 7. To work in 3/4 or 5/4 means restricting the steps per loop to 12 or 15 - even coarser time grids.

Modern Performance Sequencers - Elektron Octatrack

Modern performance sequencers mostly retain something like the above timing restrictions, though some allow for finer grids. Elektron's Octatrack[16] is a particularly advanced example among performance oriented sequencers, allowing timing offsets to be applied to individual notes. This allows for some subtle features of human rhythm variation, such as expressive anticipation, and delay of the beat from one bar to the next, to be specified.

In addition to this Octatrack allows users to work in Roads' second rhythmic space, by supporting the use multiple loops, each with different step lengths. This allows for fast and (once learned) intuitive specification of polyrhythms of the kinds heard in various world folk musics.

These advances in rhythmic nuance still do not get us out of Roads' first two spaces, however. Also, as complexity increases we rapidly return to the problem identified by Roads, with detailed control limiting immediacy of expression. Microtiming detail must be specified by conscious intent where it might ordinarily arise from muscular training and musical instinct. This splits the performer's focus between detail and overall musical effect. Also, the control of the parameters may also be too finicky for fast, improvisatory exploration. Finally this kind of intentional control of nuance still limits the possibility of serendipitous "error", and capitalisation on that "error" - which is so much a part of improvisatory playing on acoustic instruments.

Algorithmic Performance Sequencers

One popular response to the problems outlined above is to use a somewhat algorithmic approach to music sequencing. In this case some details of the music are generated by a

computer algorithm. The algorithm here may be highly autonomous or quite constrained by user controls, and the musical properties controlled may be few, or comprehensive. Neurythmic fits into this category, taking an algorithmic approach which still allows considerable control over detail.

Analog Modular Sequencing

One approach to algorithmic performance sequencing which is currently undergoing a popular resurgence is analog modular sequencing [14]. Here, analog waveforms and clocks drive sequencing grids not dissimilar to those described above, with users specifying and tweaking signal flow controlling the underlying clocks and triggers.

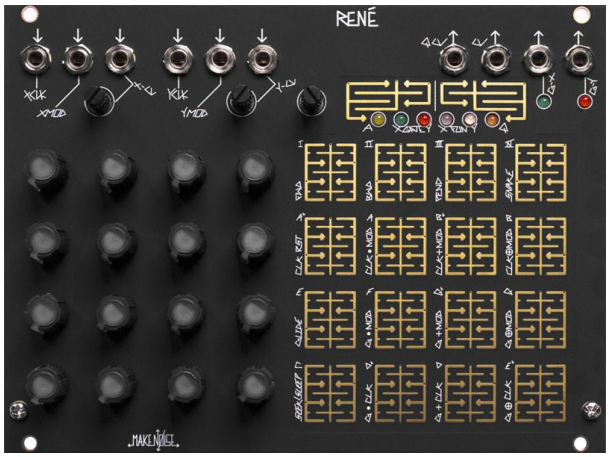

Figure 4.1: The analog rene sequencer by makenoise [17]

At its most basic, analog modular sequencing is functionally identical to the "classic" step-sequencer approach described above. However opportunities for routing, modifying and performing logic-operations on the underlying control-signals pushes this into algorithmic territory allowing programatic deformation and even randomisation of sequencer timing grids. As a result, the outputs of such systems often stretch into areas of considerable rhythmic complexity - potentially taking in all of Roads' rhythmic spaces.

Though the model was developed on analog electronics, it is increasingly common to see the same approach taken in software, using music environments such as reaktor, pure data and Max/MSP which provide elementary audio-signal building blocks.

While incredibly flexible it relies on the user to build her own sequencer. This allows for huge creativity and may result in an interface precisely tuned for intuitive performance in the user's own style, but also requires considerable understanding of signal flow and signal processing concepts, as well as an outlay in development time.

Euclidean Sequencers

Another approach, also popular in recent years is "Euclidean sequencing". Euclidean rhythm sequencers are based on the approach outlined in [18], where the Euclidean algorithm for computing the greatest common divisor of two integers is shown to be a compact method for generating a large range of traditional music rhythms.

In essence, this is an algorithm for distributing a discrete number of beats as evenly as possible over a discrete number of steps. These patterns loop in time and so are often represented as circles, which Toussaint calls "rhythm necklaces". Toussaint shows that the approach approximates a large number of rhythmic patterns common in folk musics.

There are a huge range of implementations of this approach, from visual interfaces which incorporate step-sequencer-like elements, to bare-bones functional implementations of the algorithm in live-coding languages, like the Haskell-based TidalCycles [19].

All implementations I have seen support the offsetting of cycle phase, and allow for multiple simultaneous "rhythm necklaces". This, combined with compact specification of

rhythms via a pair of integers, makes the the system, once learnt, incredibly well suited to the fast specification of quite complex polyrhythms (e.g. Roads' second rhythmic space). In "live-coding" situations in particular ([19]) this allows for the transformation of overall rhythmic pattern with only a few key strokes.

The trade-off however is that it is not possible to specify arbitrary rhythms with the system, only the equally spaced "Euclidean" rhythms described above In addition, the system does not, escape Roads' first two rhythmic spaces, and unlike Neurhythmic, cannot not generate subtle microrhythmic variation.

Synkapater

The Synkapater system is an approach to sequencing which primarily targets Roads' second rhythmic space. It uses a looping step sequencer approach, but allows for multiple simultaneous loops of different step lengths [20]. It shares the limitations discussed for Euclidean systems, above, and lacks anything similar to their compact 2-integer specification of rhythm. In place of that the system allows for arbitrary note placement.

Gears

Gears presents a novel, playful approach to algorithmic, interactive sequencing[21]. The system takes an intentionally game-like approach, with a puzzle-like rhythmic control system based on mechanical gears. Step sequencers like those described above can be attached to gears. As the gears rotate and the attached rhythmic sequences pass a cursor, they play. Gears of different sizes can be used to drive one-another, encouraging the use of multiple rhythmic loops running at different rates to create polyrhythms.

Although not supporting the level of complexity and flexibility on display in analog modular systems, this system allows generation of rhythms in most of Roads' rhythmic spaces. Unlike Neurythmic it does not appear possible to control the overall rhythmic behaviour of the system with single gestures, nor does the system generate subtle rhythmic variation in like-patterns over a number of repetitions.

Chapter 5
Digital Instrument Design

Instrument design is a particular subclass of HCI, and its concerns overlap with mainstream HCI in many ways. For example, the user centred design approach commonly used for developing interactive systems [22] is useful in both instrument design and broader HCI - and a version of that approach is taken in this project. In some areas, however the problems of instrument design can be a little more specific. This chapter reviews relevant literature on digital instrument design, focusing on the issues particular to this area.

Many HCI objects solve clearly definable and measurable problems giving reasonably clear criteria against which the success of the object can in principle be measured. Instruments, particularly new instrument designs, are not like this - their outputs are aesthetic and the range of successful output values and measures are harder to define. As Andersen and Gibson point out, in place of measures such as efficiency of task execution, instrument design targets qualitative metrics such as "intuitive modes of expression, and unbroken periods of concentration within the experience of playing music"[23].

Composed Instruments

Where mainstream HCI often focuses on improvements measured against existing, well-defined tasks, an increasing body of literature acknowledges the part that instruments themselves plays in the definition of musical aesthetic standards, and so of the musical task.

Schnell and Battier make the case that digital instruments are fundamentally different from established, traditional instruments, partly in the way they break the barrier between composition and instrument building [24]. In particular they focus on instruments which (like Neurythmic) break the direct link between the musican's gesture and sound-production. They suggest that such instruments share something with compositions - "composed instruments" - in that they define the range of possible musical structures in a way that, say, a violin does not.

Johnston picks up these thoughts and places them in a more traditional HCI context. He notes that the design of objects defines affordances and features which shape the behaviour of users[25]. In the case of musical instruments, this shaping of behaviour always involves restrictions of creativity. He points out that because of this, it becomes a significant question who is involved in the definition of a digital instrument, and how much can users appropriate and reconfigure it to their own ends?

Restriction is of course a neccessary and inevitable part of instrument design - a piano can only play discrete intervals. But pianos are often been retuned [26], or otherwise altered ([27]). This kind of alteration is not as easily available to most users in the case of software, and so restriction must be considered carefully.

Insofar as restriction cannot be avoided, the vision of musician end-users must provide an important criteria - perhaps more so even than in ordinary user-driven development. Andersen and Gibson offer a model for this, describing their instrument development process, working closely with a particular performer, aiming to create structures that "might fit better with the artists' mental images of the music" [23]. They describe a process of working closely with a particular musician, allowing that musician to define the agenda for the design, and describe practices which foster that approach.

Instrumentness

Instrumentness is a concept developed by HCI researcher Olaf Bertelsen. It aims to describe and understand the particular qualities of instruments in order to inform design of instruments, and of other, non-musical interfaces.([28],[29], [30]). As well as picking up on many of the points discussed above, it develops certain concepts which I found instructive in the design of Neurythmic.

Clusters of Artefacts.

The working practices of musicians interviewed by Bertelsen consisted, in part, in building their own music creation systems. This did not neccessarily mean developing systems from scratch, but reconfiguring and connecting together software and hardware components of different kinds[28],[29]. He argues that this practice is important to the capacity to reappropriate and creatively misuse tools outside and against their designed-in restrictions. While the current Neurythmic design does not include connectivity to other systems, since this feature was not useful in testing, this is high on the list of future developments. Further, the development of the MNO object for MAX/MSP discussed in ch. 6 supports this kind of musical activity.

Metonymy

Bertelsen argues for metonymy as a principle for interface design in instruments. Metonymy in this context is a device based on contiguity and materiality. A classic illustrative example of metonymy is the phrase "the pen is mightier than the sword", where the pen is able to stand for the written word and the sword for military aggression through their respective material participation in these spheres.

Metonomy stands in contrast to the more familiar device of "metaphor". Both devices can participate in design as much as literature - the "desktop" trope in OS design is a metaphor, and music systems often use metaphors such as tape-recorder style controls for digital audio recording.

Bertelsen argues that metaphors often stands between users and their understanding of material processes. I have found it useful to understand thi sby analogy to the problem of leaky abstraction in software development - where simplifying abstractions presented in APIs are problematically violated in certain circumstances. In the case of interface design the issue is perhaps more serious since users are often less technically empowered than developers, and less able to "get around" the abstraction.

By contrast, Bertelsen argues that metonymic interface design, based on contiguity and materiality, better exposes the mechanics of the system to the user, allowing creative exploitation in ways that the designer could not anticipate.

In the case of Neurythmic a hybrid design strategy has been applied, making use of both metonymy and metaphor, as discussed in (see sec.10)

Part II

Analysis and Prototyping

Chapter 6
Prototyping and Model Building

Overview

In this section I present initial sketches, prototyping and development for the project.

The focus of this project on aesthetic outcomes and interaction, and the lack of previous models for CPGs meant an empirical, and user-centric approach to research seemed most appropriate. In this prototyping stage I followed the pattern of the larger project by incorporating user-feedback whenever possible. Only here due to the faster, exploratory nature of development, I did so more informally and opportunistically, grabbing friends and colleagues when they were available. Regardless of its informality this approach was very useful and I am grateful to all who helped in this way, and whose insight I was thus able to incorporate.

As discussed in the previous section, prototyping was carried out in MAX/MSP - a high-level music development environment, which allows fast development of musical and interaction ideas. To work in this way I first had to develop a library which allowed simulation and control of Matsuoka's Neural Oscillator(MNO) nodes in MAX/MSP.

Matsuoka's Neural Oscillator Library for MAX/MSP

To support prototyping in MAX/MSP, I developed an extension object for MAX/MSP allowing simulation and control of MNO. This was coded in C using the MAX 6.1 API. Coding in C meant that this same code could be reused in the C++ CPG engine I would later develop. Below I discuss the implementation of the MNO calculations, reused in the main engine, and other features specific to the MAX/MSP external, which as discussed above, I intend to make available to other users to enable further research in this area.

Computation of Matsuoka's Neural Oscillator

MNO is a set of ordinary differential equations. To compute this in discrete steps, I used the 4th order Runge Kutta method. Runge Kutta was chosen since it offers increased accuracy over Euler's method.

My method was to calculate spectral coherence (Using George Moody's implementation of the spectral coherence function[31]) between the same function calculated at step size n and n/2, repeating this for various values of n. Improvements in spectral coherence tailed off after stepsize 0.25 and I chose that as my default step size, but have exposed stepsize to the user through the application interface.

Variable Step Rate and Interpolation

To support easy control of oscillator frequency independently of equation parameters (and avoiding the need to put in place systems to use Matsuoka's equation for approximating the frequency of the oscillator[12] where precision of operation is not required) I implemented variable calculation step rate.

Variable step rate is achieved by means of a variable rate phase ramp which triggers calculation of the next step on phase wrap. Since this approach divorces MNO step-rate from audio sample rate, interpolation is necessary. I provide three interpolation methods to address three specific use cases.

- "drop-sample" interpolation for efficient calculation in non-audio situations - the MNO output value is simply held until the next MNO calculation.
  Computationally very cheap but spectrally distorting, introducing considerable aliasing noise (harmonic partials with considerable energy above nyquist at even low root frequencies). Only suitable where sample-rate = calculation rate.
- B-spline interpolation
  More computationally expensive, but aliasing noise is reduced considerably. Suitable for audio rate applications, such as the direct synthesis of sound.
- linear interpolation for cases.
  A compromise between the other two options, still introducing considerable aliasing noise, but its continuity makes it suitable for low frequency control applications such as parameter modulation.

B-spline interpolation was chosen for audio-rate uses on the basis of evidence in Oli Niemitalo's well known "pink elephant" paper[32] which analyses the signal-noise-ratio and instruction count of a range of audio interpolators. B-spline interpolation offers the best signal-noise-ratio of any non-oversampling approach described there.

Stability

To improve usability I constrained parameters to ranges in which the oscillator remains stable (see fig 3.3).

Prototyping and Testing Framework

The MAX/MSP object discussed above was then used to develop a system for hands-on testing of musical interaction schemes. A screenshot of this is shown in 6.1

This framework allowed me to me quickly set and alter the topology of an arbitrarily sized MNO network by connecting virtual "wires" between nodes. It also allowed me to test different ways of sonifying the network output, devise different approaches to generating rhythmic triggers, test network topologies, control curves and test the effect of external inputs to the system, such as noise, sine-waves, etc. In later stages of development, I used this framework to prototype and confirm control logic before implementing it in C++ code.

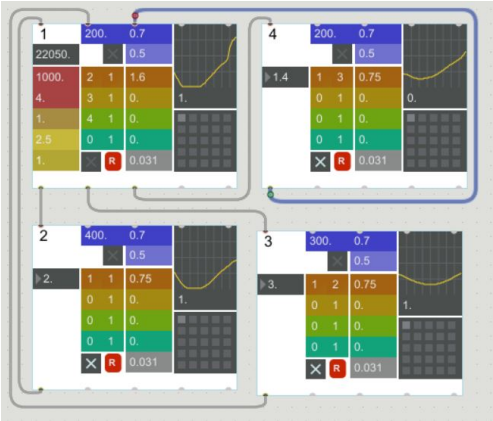

Figure 6.1: Matsuoka Oscillator Rhythm Testing Framework in MAX/MSP. Wires show interconnections, input weights are set by values in col 3, rows 3-6. Nodes 2,3,4 are slaved to parameters of 1, and their "natural frequencies" set as multiple of parent's

Prototyping Outcomes

Control of Node Frequency

As discussed previously, Matsuoka provides a result in [12] allowing the approximation of the frequency of an unconnected half-centre MNO, from its equation parameters. I found that this approximation was not accurate enough for directly control of frequency in my application, resulting in tempo drift. To address this, I adapted Matsuoka's result slightly to suit my purposes, and developed a calibration technique which considerably improved accuracy.

From the original equation (fig.3.4) we can see that it is possible to control approximate frequency by varying only the equation parameters τ_1 & τ_2, and maintaining the proportion between these values. This is significant in terms of reducing complexity of tuning, and interaction, because wave-shape and amplitude remain constant under these conditions - both factors which influence entrainment behaviour [12]. Following from this, I adapt the equation to express τ_2 as a multiple M of τ_1. I also incorporate a constant E for the approximation error, and another, S for the sample rate of the simulation.

$$\tau_1 \approx \frac{CS}{\omega}\sqrt{\frac{(\beta+\beta M-M\gamma}{M\gamma}} \text{ (where } M=\frac{\tau_2}{\tau_1},\ S=\text{sample rate},\ E=\text{error constant)} \quad (6.1)$$

Figure 6.2: Improved approximation of τ_1 for 2 neuron Matsuoka Oscillator

Experimentation revealed that if frequency is controlled in this way, by changing only τ_1, then a single value for the error constant E suffices for the entire range of operation useful for rhythm generation (cycles from ˜0.05 to ˜20hz, and in fact comfortably beyond). This is significant since if the other equation parameters are tuned in advance it allows the error constant to be measured and set ahead of runtime. Failing this, if control of wave-shape is required, then a calibration procedure can easily be implemented which runs when the application is loaded, or when wave-shape is changed.

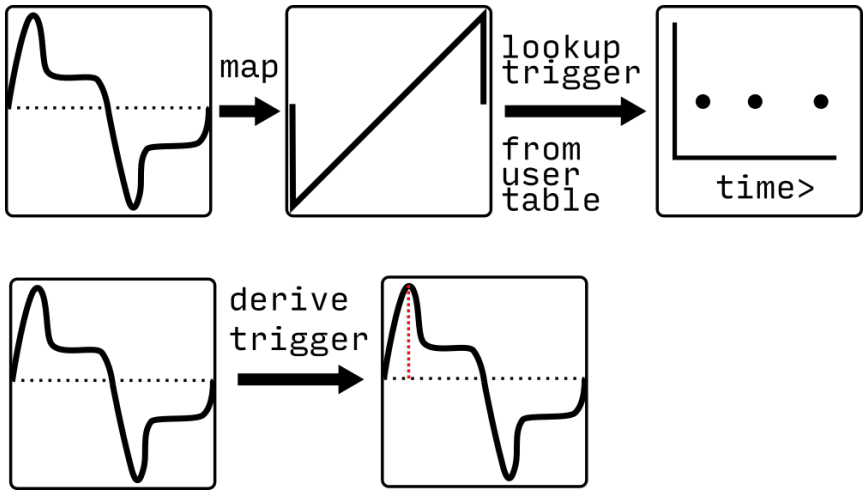

Figure 6.3: Two schemes for rhythm triggering. top: derive a phase ramp from the output waveform, use that to read a table of note positions defined by the user
bottom: derive a single note trigger from signal peak, zero crossing, etc.

The calibration procedure itself is quite simple. An isolated node is simulated at 1hz for a number of cycles, and its actual frequency over this period (measured by count of positive zero crossings) is compared to the estimated frequency from the equation above. Accuracy can be improved by increasing sample rate. The error constant can then be derived from the equation.

Phase Alignment

Since the approach I am taking generates rhythms from the phase patterns of a network, it is naturally often useful to be able to reset phase relationships directly - when adding a new node, for example, it is desirable that it begins synchronised to the others, even in the absence of connections to other nodes.

I tackled this problem by taking snapshots of nodes' internal state at relevant moments. Since I ultimately opted to use the same equation parameters for all oscillators (see 6) I was able to record internal state at positive zero-crossing for an oscillator tuned this way, and use that to reset any oscillator in the system to the same state to the start of its cycle.

Similarly, when implementing presets I ensured accurate recall of network behaviour by recording and re-applying the internal state of all oscillators in the system.

Sonification

One major question to be answered was how the network should map network behaviour to sound output. As discussed in chapter 3, the only previous research into musical uses of CPGs focused on direct playback of the signal outputs of the network, so a model for rhythmic sonification had to be created from the ground up.

I arrived at two potential schemes for sonification of the network. The simpler of the two was to generate a note-trigger from the signal output once per cycle - a model common in electronic music. The second option was to derive a phase ramp from the output signal, using the technique discussed in [33] and use that ramp to drive a conventional sequencer grid, on which users could specify rhythms. These two are illustrated in fig 6.3. Though

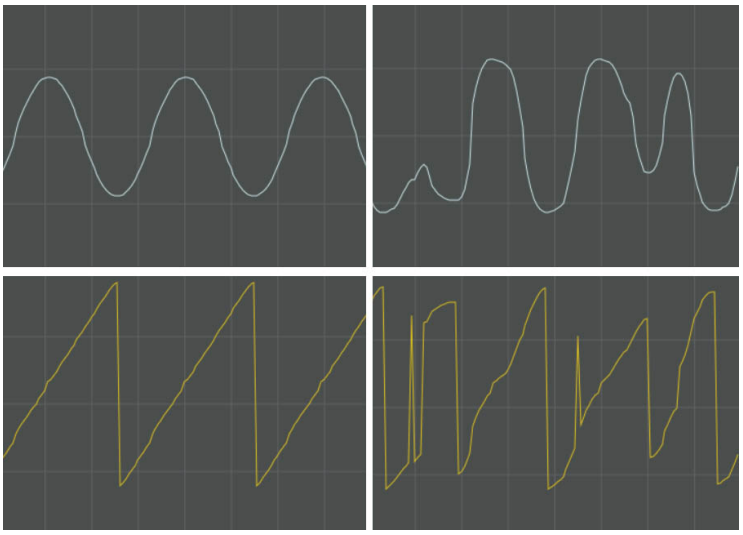

Figure 6.4: Technique in [33] for mapping output to phase works well for outputs which approximate a sinewave (left), but not for the outputs of more complex networks

the latter option seemed promising, and had the virtue of bringing the system closer to familiar grid-based sequencing approaches, I found problems with it.

Issues with Phase Ramp Approach

I found the phase ramp derivation too unreliable for my own application. The process described in [33] assumes the wave-shape of the oscillator to be close to that of a sine wave. It is quite possible to tune single oscillators to give such an output, but even so I found the mapping quite fragile for complex networks when wave-shape was distorted by entrainment effects.

It may be an interesting area for future research to develop musical tools with simple networks, using this phase mapping approach. These might use CPGs for force-feedback control of rhythms for example, and add musical complexity by other means. However, in developing this project, I wanted to focus on effects coming primarily from the network, to allow a deeper evaluation of the specific properties of CPG rhythm generation for musical creativity.

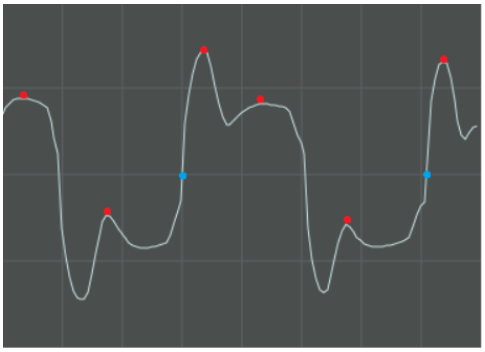

Figure 6.5: Many-to-one input with moderate signal strength invalidates local-maxima (red) triggering, but positive zero-crossing still gives one trigger per cycle.

Triggering Schemes

Having decided to generate rhythms using triggers derived directly from the CPG waveform, it was necessary to decide on a triggering scheme. I started by trying two simple approaches, familiar to me as well trodden techniques in electronic music:

- trigger on zero crossing,
- trigger on local maxima.

Of these two approaches I initially found zero-crossing to be more effective. Where signal connections are moderately strong and many-to-one, a node's signal might have several local maxima per cycle. Although it is also possible for zero crossings to proliferate at extremes of connection weight, I found this was far rarer, and occurred outside what I found to be the useful range of the system.

After working with this approach for a little while I noticed that the amplitude of the peak of the waveform, seemed often to vary in a way that was related to the pulse of the incoming signal. I felt this might be used to control the amplitude of the note triggered, increasing perceived expressivity of output. To take advantage of this I implemented a comparator which triggered on the first local maxima after zero crossing. I used this comparator to trigger notes from the synthesizer engine, sending both the trigger and the momentary amplitude. I used this triggering approach through the rest of the project, including the final version of Neurythmic.

Simplifying Controls

One hurdle for intuitive interaction with CPGs is the size of the parameter space. At a minimum the system requires one weight parameter per connection, and 6 equation parameters per node. This is before adding systems to help manipulate phase, provide musical output, etc.

We will see in ch.10 that in the final application I solved the connection weight aspect of this issue by developing a graphical network interface which gave an easy-to-read visual summary of network weights. And above in this chapter (sec.6) we have seen that I was able to implement a system to control node frequency by a single parameter, based on a result in 3.

Another fundamental control factor when working with rhythm is time-spacing between notes. In the present case that means, in part, phase offsets between oscillators. I found that in robotics applications, this seemed to be tackled either by tuning the parameters of each individual oscillator to create the desired overall effect, or by using a single parameter set for all oscillators, combined with delays or phase mappings at node connections [33]. Tuning individual oscillator parameters was obviously far too complex an interaction method, and I had already ruled out the phase mapping approach (sec.6), so I took the approach of adding variable time delays. This is discussed further below.

I found a quite simple approach to this problem - simply adding time delays to the system - either on the connections between nodes, or between nodes and their musical output. This is discussed further in the next section. With these control simplifications in place I found that, with practice, I was to be able to control the systems and generate musical patterns. Much of the rest of this project describes the refinement of these control mechanisms and interfaces to them.

Signal Offset

In the section immediately above this I discuss the use of delay to specify the timing offsets between one node and another. I tested two locations for this signal offset

1. Immediately before note-trigger generation
2. Immediately before input to a node

Of these, delaying before trigger generation seemed to me the most understandable of the two controls - the effect was immediately audible and exactly as specified. I found that particularly if this control was scaled to a range from zero to the length of the parent input's cycle, I could easily use this control move a voice's pattern precisely relative to its parent.

Delaying before input to the node was a more complex control. It was often harder to predict the result, since its effect was dependent on connection weight, and this effect could be lost if connection weight was reduced. Unlike the output delay, however, delay in the connection affected ongoing connections in the network. It was also useful when working with feedback relationships, and I found I could move the system from lock-step rhythms into rhythms that evolved over time by manually tweaking the delay. Though harder to use, I found the quality of discovery using this feature quite satisfying, and resolved to include both options in the user test build, allowing users feedback to influence the final design.

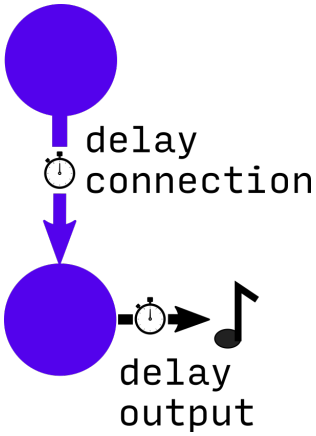

Figure 6.6: Two potential locations for signal offset, for control of note placement

Despite my impressions of the relative merits of two controls, during informal pilot tests of an early version of Neurythmic I found that users were confused by the distinction of between the two offsets, and did not understand why the pre-output version only affected the node in question, not connected nodes. As such I removed this from the first version of Neurythmic, but added it back into the system in the final version when I was able to find an explanatory context for it.

Network Hierarchy

Some of the literature on Matsuokas Oscillator use in robotics (e.g. [33]) describes coordination from a central oscillator as a simplifying strategy for network specification. Similarly in my own prototyping I found it a useful principle to begin by using a branching "snowflake" structure, with branches radiating from a central node, and only later to add connections between branches. Using this structure encouraged the creation of rhythmic subgroups, nested beneath cycling root patterns, a common form of organisation in electronic music. Without this kind of organisation I found that networks could quickly confusing to read.

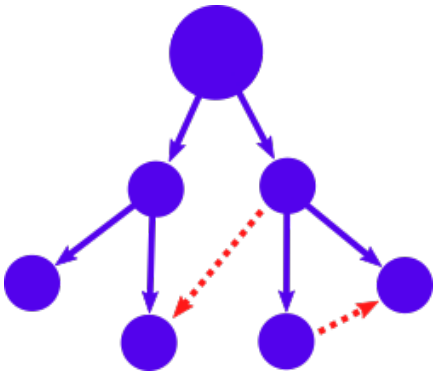

Figure 6.7: Beginning with branching structures first (blue) and only adding cross connections afterwards (red) helps keep track of relationships and manage rhythmic coherence

I devised a means of encouraging use of this pattern by making a formal difference in the interface between "parent-child" connections, and "ordinary" connections. In this scheme, the "parent-child" connections have a control-organisation functionm as well as carrying signal, with the frequency of child nodes set as multiples of their parent's frequency.

While appearing useful when prototyping, we will see later that this mode of organisation was not entirely successful in user testing, and was dropped from the final design.

Chapter 7

Analysis and Tuning of the Network

Introduction

In this chapter I present results of simulation and analysis of MNO, responding to issues found in user testing. Some results here feed into further development, others confirmed that system behaviour can be expected to remain within acceptable bounds. Many of these observations may be of wider interest for musical and creative applications of MNO, including. In particular I provide an equation for deriving the stopping threshold of Matsuoka's Neural Oscillator, and descriptions of rhythmic behaviour of the system with inputs below the entrainment threshold - a resource perhaps not useful outside creative applications and, based on my search, apparently not found elsewhere.

I note in the introduction that my approach to development called for me to get a working system into the hands of users as soon as I could. Guided by the desire to work quickly, the early stages of the project, up to the first user prototype were carried out using only results from the literature, and hands-on tuning by progressive approximation in my MAX/MSP prototyping rig (see 6). For context for this approach, see discussion of "tuning by feel" in 3).

Working in this way I entered the second half of the project with better formulated questions, following from formative user testing. Some of these questions implied an answer via simulation and analysis of MNO. The process of answering those questions is discussed in this chapter.

In broad overview, I developed programs in C++ to simulate results for a wide range of network setups, under different conditions of use, and then performed analysis on the results using the Pandas framework in Python. These simulation scenarios were designed to answer specific questions which arose from user testing and my own hands-on experience.

I will first lay out the details of implementation of these tests, before discussing the test results in the context of the questions they aimed to answer.

Implemention and Analysis of the Simulation Tests

Tests were implemented in C++, using the same engine written to power my application (see 10). Tests were run in the debug mode in Visual Studio, allowing me to work more interactively, taking advantage of the ability of the system to recompile while in break state, then resume operation. Given the scale of calculation involve (some runs taking hours), this interactive approach allowed me to observe and debug behaviour over a few cycles of simulation before letting it run in release mode for speed.

The results were output to CSV files and analysed with the Pandas data analysis framework in Python.

Is Final Behaviour Independent of Initial Conditions?

I noted in the previous chapter that initial conditions could be decisive on the final rhythmic behaviour displayed by a network. (see e.g. 6). Some issues around this were easily solved (e.g. presets could include system state), and we will see later that users in fact valued a degree of unpredictability in the generative behaviour of the system. Nonetheless it was important to understand the bounds within which the system could be expected to be predictable. I therefore set out to investigate the dependence of final settled states, on initial conditions. Finding that final conditions are stable, the results below did not directly impact on later system design, but proved useful later in the project as descriptive guides to musical behaviour.

I began these investigations focused on the simplest case - a two node, feed-forward network.

Method

I wrote a program to simulate these networks and record the timing of positive zero crossings for both oscillators. The goal was to analyse the phase relationship over time between the two oscillators, under various conditions. The equation parameters used were $t_1/t_2 = 4$, $b = g = 4.07$, $c = 1$).

Oscillators were all initialised with an internal state taken at zero crossing from an oscillators running at 1hz with no inputs and the same equation parameters. Both oscillators were tuned to the desired frequencies, then run for 16 cycles, unconnected, halting on a positive zero crossing of the parent node. This aimed to allow any influence of initial conditions to be minimised.

The child node was then moved into the desired initial phase relationship by first simulating the node until it reached it's positive zero crossing, and then simulating another $pw/2\pi$ steps - where p is the desired phase value in radians and w the wavelength of the node in samples.

Finally the whole system was simulated and the timings of positive zero crossings for both nodes recorded until either 512 cycles of the parent node, or 128 cycles of the child node had passed, whichever came first. This procedure was repeated for all combinations of

- starting phase relationships between 0 and 1.9π radians, rising in intervals of 0.1π
- frequency ratios of child:parent at values between 0.5 to 7
- input weight to the child node at values between 0 and 8.

These values were chosen to encompass the predicted operating range of my program based on experience gathered in the first phase of development.

Results

Plots

Figures 7.1, 7.2 and 7.3, plot the phase relationships that occured during simulation, with phase recorded at each positive zero crossing. The results aggregate simulations for initial phase relationships varying from 0 to 1.9π in steps of 0.1π. Results are shown 100 cycles into the simulation described above, to show final, rather than transitional results. Frequency of recurrence of a value is shown by the size of the marker, so that larger markers indicate that a phase recurs with great frequency, and small markers show rarely occuring phases.

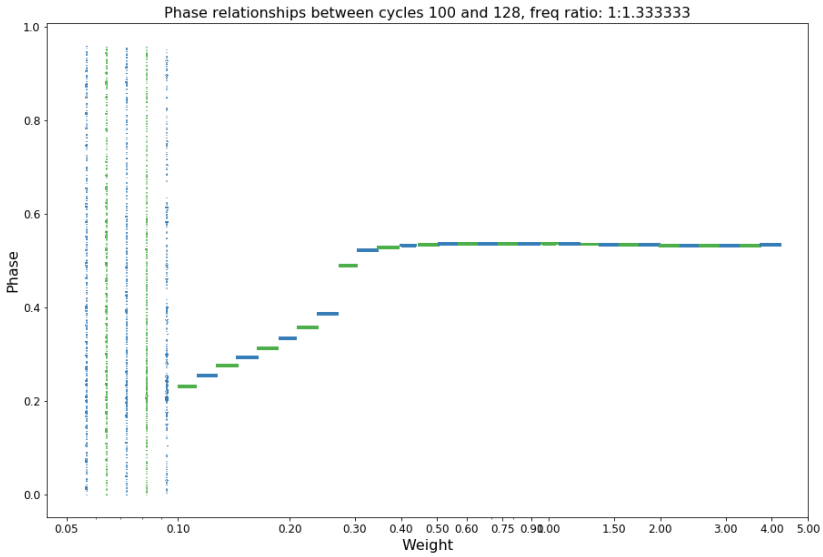

Figure 7.1: Phase relationships for pair of oscillators as connection weight from one to the other increases

When nodes are entrained, their final phase pattern is independent of initial conditions

Unsurprisingly, the tendency of the system to converge from diverse starting conditions to closely similar final phase relationships (or sequences thereof) depended primarily on input weight. Above the weight threshold at which entrainment was achieved, and for a given frequency ratio between the nodes, networks converged strongly towards a single phase relationship. This effect was seen regardless of starting conditions. This strong entrainment condition was achieved for all frequency ratios tested.

Weight values above the entrainment threshold give no interesting results

An incidental but useful result worth observing here is that once the threshold for entrainment has been reached, further increasing the weight has very little effect. Beyond this threshold we see only a small adjustment to the phase relationship, and nothing beyond this. Since the adjustment is small this is not a promising means of controlling phase.

Final response to input weight is highly consistent, and divides into distinct regions, with different musical effects

Another interesting result is that rhythmic behaviour moves through distinct regions as weight is increased. This is best evidenced on the plots for freq ratios 1:1.7142 (1:$\frac{11}{7}$) and particularly 1:5 (7.2 & 7.3). In the latter in particular we can see 4 very clearly defined regions, examination of which is illustrative.

Region 1 At low weights, the child node retains the signature of its natural frequency. The cycle slows slightly as weight increases, and at a certain, quite low input level (e.g. above 0.07 in fig 7.2), the oscillation begins to "reset" to the parent node's cycle such that the whole pattern (perhaps over a number of parent cycles) repeats reliably

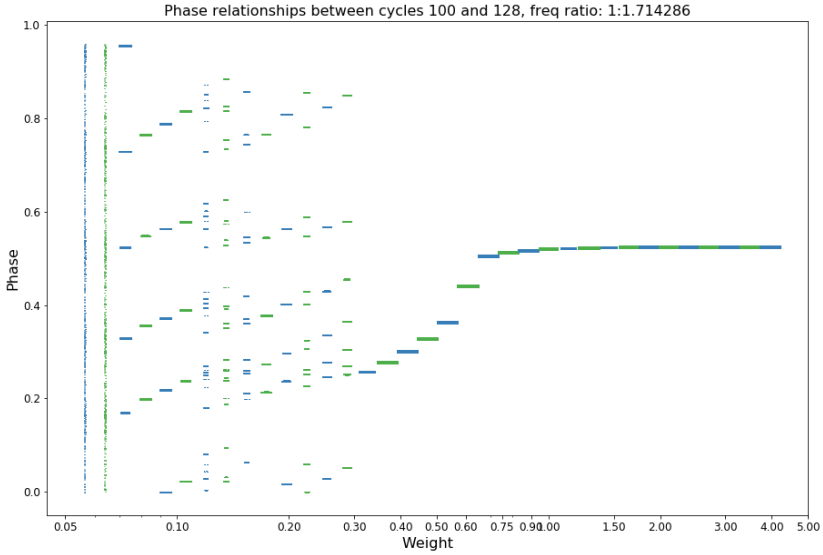

Figure 7.2: Phase relationships for pair of oscillators as connection weight from one to the other increases

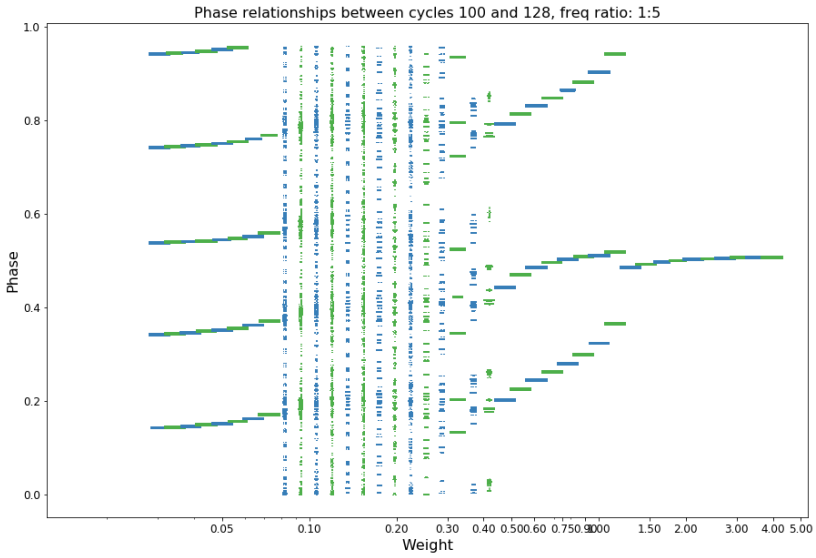

Figure 7.3: Phase relationships for pair of oscillators as connection weight from one to the other increases

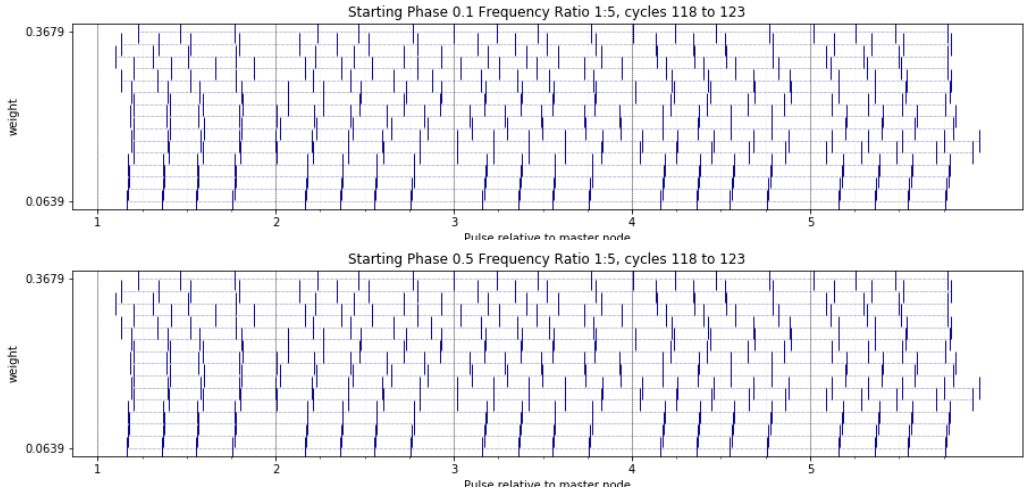

Figure 7.4: Rhythms in the "noisy" region of 7.3 are due to variation between consecutive cycles, and are still independent of initial conditions

each time. This lengthening-and-resetting pattern shows great consistency, with no variation due to different starting phases. While the figures shows cycles 100-128 after initialisation, the pattern is actually established quickly - within a couple of cycles for all but the lowest weight values, and so can be relied upon for realtime user interaction.

Region 2 Above this, a "noisier" region is seen, with much greater variation in phase patterns. At first there is still some clustering around the trajectories plotted by the first region, but this soon disappears. Looking at the data in this region in more detail, and in the context of its time sequencing (fig. 7.4) we can see that this "noise" does not come from variation due to initial conditions. The "noise" in our plot is rather due to cycle-to-cycle variation patterns in the child node's phase, with patterns taking two or more cycles of the parent node to repeat. In my experience this is heard, depending on conditions and context as either arhythmia, or as an attractive and unusual rhythmic variation - in any case more complex patterns than are found in the region below.

Region 3 Above this we see another region, or arguably two. First a short transition where the networks still show variability, but with far stronger clustering around particular phases, followed by another range similar to 1) above, where behaviour again becomes very regular. In fig.7.3 we see three cycles of the child node for every cycle of the parent node. As in the first region, these phase relationships repeat consistently from cycle to cycle, and we see a gradual offseting of phases as weight increasesl This pushes the zero crossings of the first and third cycles progressively later in the parent node's cycle. In contrast to the first region however, another effect begins to show itself - the second cycle shortens as the others lengthen, converging towards a phase relationship with it's input signal close to that found past the threshold for full entrainment (fourth region, below). This last result is interesting as it shows continuity within the otherwise abrupt change in behaviour between rhythmically distinct zones. This will be useful in balancing variety and confirmation of existing patterns in musical results.

Region 4 The last region is the most straightforward: full entrainment, with a single cycle of the child node for every cycle of the parent node. The phase relationship

of the nodes remains very steady for a given weight, and changes slightly as weight increases, seeming to converge towards a final value.

Stopping and Restarting the Nodes

In user testing, several users said that they would like more space in the rhythms generated by the system. In part this will be addressed in subsequent sections by improving the user interface for muting nodes, but it is also worth asking if any properties of the CPG itself could be used to increase space in the patterns generated.

Due to time-constraints, the results below were not used in the version of Neurythmic presented in this project but are presented as of potential interest to others hoping to extend this research.

While working with the networks in my first prototype system in MAX/MSP (see 6) I found that a signal input which remained consistently above a threshold value caused the oscillator to halt. Below I present investigations into the following questions which followed from this observation

1. What is the threshold for stopping a node?
2. How quickly does the node stop?
3. How quickly does the node restart when the stopping signal is removed ?
4. How is the waveshape affected by stopping and restarting?

Equation to Predict Stopping Threshold

In this section I develop an equation to predict the stopping threshold of the oscillator. This is useful since, as we will see, the oscillator recovers more slowly from stopping if a larger stopping input is used.

This result is built on an observation made when working with the oscillator in MAX/MSP - I noticed that when stopped, it seemed always true that $x_2 > 0$ and $x_1 = 0$. Though a proof of this would be preferable, I have not been able to construct one, and have continued by assuming the observation can be generalised. I do this on the basis that the result is useful and that I have been unable to find cases in which it does not hold true during testing.

In describing the result I build on this observation, it is helpful restate the equations of the oscillator

$$\tau_1 \dot{x}_1 = c - x_1 - \beta v_1 - \gamma [x_2]^+ - \sum_j h_j [p_j]^+, \tag{7.1}$$

$$\tau_2 \dot{v}_1 = [x_1]^+ - v_1, \tag{7.2}$$

$$\tau_1 \dot{x}_2 = c - x_2 - \beta v_2 - \gamma [x_1]^+ - \sum_j h_j [p_j]^-, \tag{7.3}$$

$$\tau_2 \dot{v}_2 = [x_2]^+ - v_2, \tag{7.4}$$

$$y_{out} = [x_1]^+ - [x_2]^+. \tag{7.5}$$

Looking at these equations, it is clear that, when the signal is stopped, $\dot{x} = 0$ and $v \to x$.

Following from the latter observation we simplify by substituting a new variable z for both v and x. Together with 7.1, this gives us

$$0 = c - z - \beta z \tag{7.6}$$

or

$$z = \frac{c}{\beta + 1} \tag{7.7}$$

Similarly, from 7.3, and substituting T (threshold) for the summed input $\sum_j h_j[p_j]^+$, we get

$$0 = c - \gamma z - T \tag{7.8}$$

from here, we can substitute in 7.7 for z and rearrange to give us:

$$T = c - \frac{\gamma c}{1 + \beta} \tag{7.9}$$

which allows us to derive the threshold for a static signal to stop the oscillator, given a known tuning of the oscillator.

Behaviour on Stopping

I extended the tests outlined in 7 to address the remaining questions. This time single nodes were used, taking the same approach to initialising and stabilising the node. I began with no signal input, then introduced a static stopping signal, whose value derived from result 7.9 and recorded the output. I repeated this, changing the point in the node's cycle at which the stopping input was introduced.

I then performed another set of tests, beginning with a static stopping signal and a stopped node, and removing the signal to record the behaviour of the node on restarting. I repeated this for various values of b and g, with $b = g$.

Results

Figure 7.5 shows the results of the stopping test described above. The figure superimposes multiple simulation results with the stopping input introduced at different points in the oscillator's cycle - the markers indicating that point. With a static input at or above the stopping threshold described in 7.9, the oscillator always halts before its next positive zero crossing, settling at a value below zero. This result was consistent across all conditions measured. This is a useful when triggering events at the positive peak of a node's output since the effect of a stopping signal on the rhythms generated will be instantaneous.

Removing the stopping signal caused the oscillator to immediately swing upwards, resuming its cycle very close to the beginning. I measured the delay of the first positive peak (when our system would generate a note-trigger) after the signal was removed for various values of b and g. Figure 7.6 shows these results, expressing the delay in terms of phase. We can see that higher values of b and g result in shorter delays for the creation of first trigger after the signal is resumed. Since this value is expressed as phase, we can see that the triggering delay will also be reduced as the frequency of the node stopped is increased.

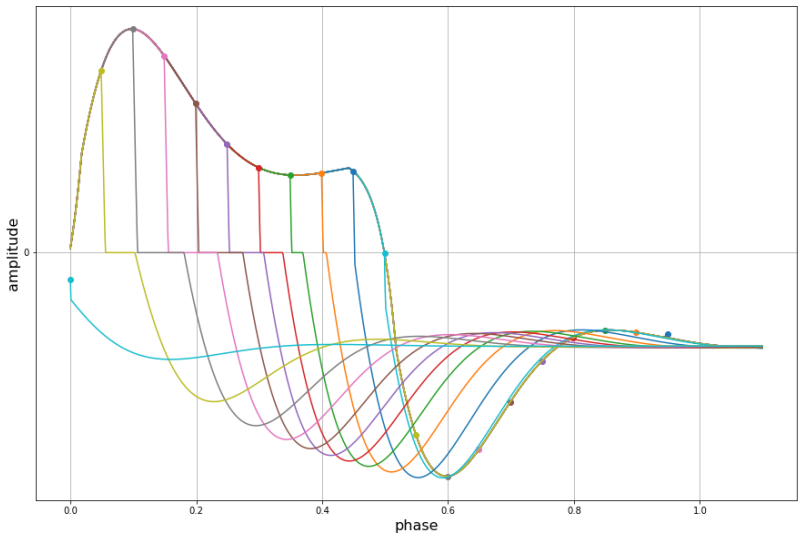

Figure 7.5: Effect of stopping signal on waveform, when introduced at various points in the oscillator's cycle

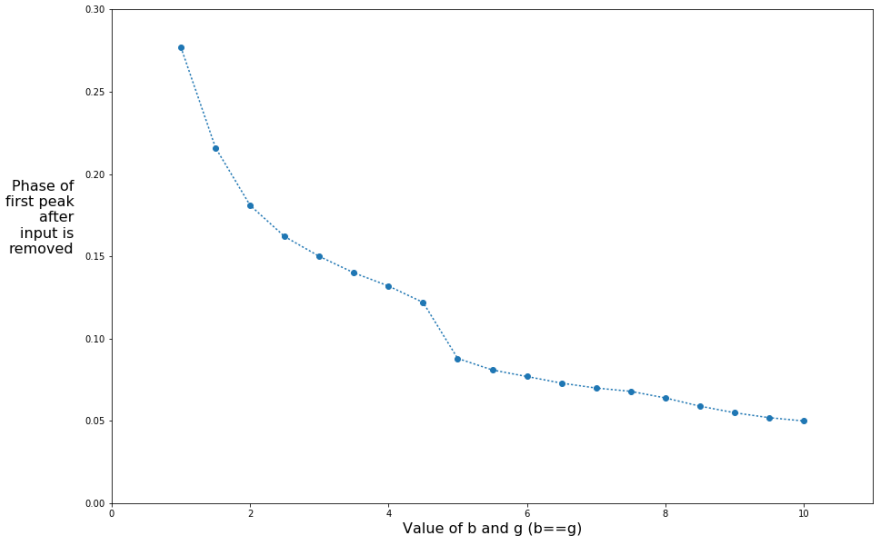

Figure 7.6: Delay between removal of stopping signal, and occurance of first positive peak reduces as b and g increase, where $b = g$

Entrainment Threshold Curves

We saw earlier that the threshold for entrainment of MNO to an incoming signal varies depending on the ratio of input signal frequency to the node's natural frequency 3.

In the first user prototype I did not scale weight controls to take this effect into account. As a result control of connection weights in the system was less predictable and onscreen feedback less useful than it might have been. During think-aloud testing of that prototype (see ch.9) users commented, for example, that changing a connection sometimes seemed to do very little.

After testing, I therefore ran simulations to generate a curve for the entrainment threshold of MNO given the equation parameters used in my system.

Definition of "Entrainment Threshold"

The first step in this was arriving at a useful definition of the entrainment threshold. Though the term is used often in the literature, and its broad meaning is fairly clear, I have not found a paper which makes its precise definition explicit. Previous papers do not, for example, indicate tolerances for consistency of the resulting phase relationship, the number of cycles "grace" given before the nodes must have acheived entrainment, and they do not quantify the degree to which the effect must be independent of the system's starting phase relationship. It is possible that these questions are not significant in previous use cases, but in the current case these did seem to me significant questions, affecting musical outcome and user experience.

I therefore arrived at a definition of entrainment, suited to my own use case.

- behaviour must remain consistent over a period of 16 cycles
- within this period the child node must exhibit a single positive zero-crossing for each positive zero-crossing of the parent node.
- "consistency" is defined by the requirement that the standard deviation of the phase relationship between the nodes' zero crossings, not exceed 0.03. (This value corresponded to my own subjective impression of rhythmic regularity in the expected ranges of operation).
- a transition period of 1 cycle of the parent node is allowed, before these conditions must be met.

Method

I ran simulations of 2-node feedforward networks, varying input weight and checking for entrainment against the conditions defined above. Networks were initialised and stabilised prior to each run as described in section 7 above, then their connected behaviour simulated, testing against the conditions above.

Entrainment threshold was found each time using a binary search between limits a listing of the code for this binary search is provided in appendix D.

The frequency of the parent node was kept static at 1 cycle per second, while the frequency of the child node was increased by steps from 0.2 to 8 hz. For each frequency the entrainment threshold was found for starting phase relationships between 0 and 1.9π radians in steps of 0.1π radians. The mean, upper and lower limits for stopping threshold at different phase relationships was then taken.

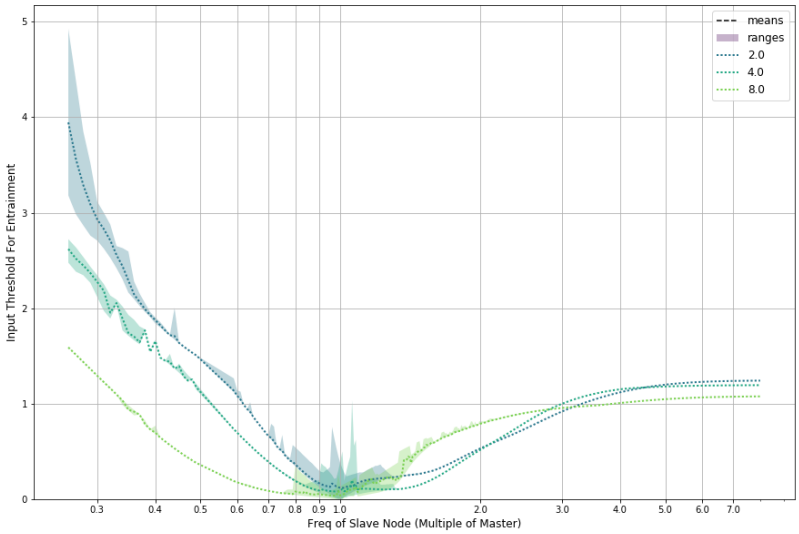

Figure 7.7: Input weight threshold for entrainment behaviour dependent on frequency ratio between nodes. Different lines plot different values of b and g ($b = g$)

I repeated this for various values of the equation constants b and g keeping $b = g$ since hands-on experience had led me to believe that the ratio between these was not a promising tuning parameter to explore.

Results

Figure 7.7 shows the mean, and upper and lower limits of these results.

Figure 7.8 shows the control curve derived for Neurythmic. This curve is used in the final application to simplify control, by scaling user inputted weight. This means that the same user input weight will always result in the same kind of behaviour from a simple connection between two nodes, regardless of the ratio of frequencies nodes involved.

This curve was adapted after trying the first results, and uses a slightly altered definition of the entrainment definition above, now allowing a transition of $0.5 < f_p/f_c < 3$ cycles, where f_p is the frequency of the parent node, f_c the frequency of the child node. This limits the time for transition to a reasonable range, while giving more slack in high child-node frequencies where an improved response would not be likely to be noticed.

Since the networks used in Neurythmic are more complex than these 2 node feedforward systems, I adjusted the recorded curve slightly to get the control curve. I took a rolling average, shifting this upwards by 0.1, and clipping it at a low value of 0.3. This adjustment gave a control curve which I judged had the right subjective "feel" based on my own interaction with the system - giving what felt like predictable and consistent results across a wide range of situations.

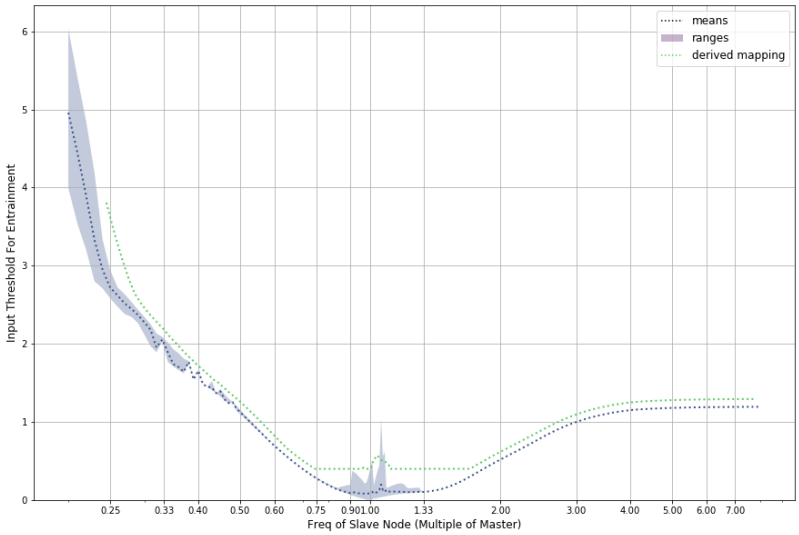

Figure 7.8: Measured entrainment threshold for equation parameters used for next stage of development, together with derived control-mapping curve for weight

Part III

Development

Chapter 8
Tools

Platform

I developed Neurythmic on Windows 10 in large part because I own a laptop with Windows 10 installed, and because all of the major development frameworks I considered worked on the OS. Working on Linux in a virtual machine did not seem an option due to the likely performance demands of the project.

openFrameworks

openFrameworks is an open source C++ library/framework for "creative coding". Built on top of the OpenGL graphics API, and bringing it together with a number of other multimedia libraries, it supports the development of high performance media rich applications, has strong documentation and an active online community. Of particular relevance to my project, openFrameworks supports cross platform development over Windows, iOS and Android. This was a consideration since the application lends itself to tablet-interaction, and I would like to port the completed system to iOS and Android.

I considered other frameworks and libraries in the same area, looking in particular at Cinder - another C++ development environment and Juce - a C++ audio development framework. All have rich multimedia libraries and support cross platform development. Juce however only has facilities for quite basic slider-centric UI design, and I expected my UI not to fit this mould. OpenFrameworks was preferred over Cinder since it seemed to have a more active online community.

MaxMSP

MaxMSP is a visual programming language-cum-environment for high-level, interactive audio development which runs on Windows and OSX. The system can be extended by writing dynamic libraries (called "objects") in C. I have considerable experience with the system and know it to be well suited to fast prototyping of musical ideas. I anticipated that the time given to implementing an MNO library for MAX/MSP would be repaid in speed of prototyping, and that I would be able to make that library available to other users for further exploration of the themes of this project.

Toolchain

Environment / Compiler
On Windows openFrameworks provides packages targeting Visual Studio or MSYS2, and I was keen to use supported approaches in order to take advantage of the knowledge available

in documentation and support forums. I opted for Visual Studio for to its out-of-the-box code-completion and refactoring facilities. Within this I used the standard Microsoft C++ compiler.

Version Control
I used Git for version control, hosting my repository with GitHub. This decision was based on my experience with the system, and the availablity of free hosting.

Documentation
I used Doxygen for code documentation. Again this decision was based on experience with the system, and the ease of setup in Visual Studio.

Chapter 9
User Centred Design

Overview

This chapter discusses the user-centred design approach taken in this project, working with four expert musicians. It outlines my reasoning for taking this approach and describes the techniques used to manage this process. The actual results of this collaboration are described in the following chapter (in the context of their formative influence on development) and in chapter 11, where the final summative evaluation of the Neurythmic system is discussed.

Development of tools in any context must take into account the end user. This observation is perhaps even truer in the case of musical instrument creation, where the tool supports creativity and may help define creative goals as well as meet them (see ch. 3 for more discussion of this).

Integrating Evaluation

Key to the success of this user centred design approach was to continually test speculation against practice, and to ensure subsequent stages of design were informed by that testing. Evaluation was required in this project to address the following issues

1. Refining the development of prototypes for interaction with CPGs towards one proposed system.
2. Describing the novel qualities of musical interaction and output possible with that system
3. Identifying potential scenarios for musical creativity with CPGs

The first of these three goals, entirely formative, and is the focus of this chapter and the next. The second two are mainly addressed by formal summative evaluation described in 11, though some results from testing described in this chapter are referred to there.

Addressing this first goal meant building evaluation and user testing into the structure of the project in a manner somewhere between Agile development and the example of [23] (see ch. 5).

From Agile I took the principles of always maintaining a working product to allow end user feedback, unit testing (using the C+++ framework Catch [34]) to reduce friction when integrating change, and regular, practice-based, face-to-face meetings with end users.[35]. The methods used to structure the last part of this are explored in the remainder of this chapter.

Collaboration with users split into two parts - informal and formal - and both were significant.

Informal Formative Testing

Informal testing was carried out with friends and colleagues, both musicians and non-musicians. This was opportunistic and structurally diverse, serving the ends of prototyping - confirming or altering design assumptions as they were in the process of being formed.

Informal testing varied from inviting people to simply play with the system and give free feedback, to presenting sketches of interface elements and paper prototypes, and asking people to interpret them (some examples of the latter are discussed in the following chapter). Though by its nature mostly undocumented and not suitable for the drawing of conclusions, this practice was very important to the way I worked.

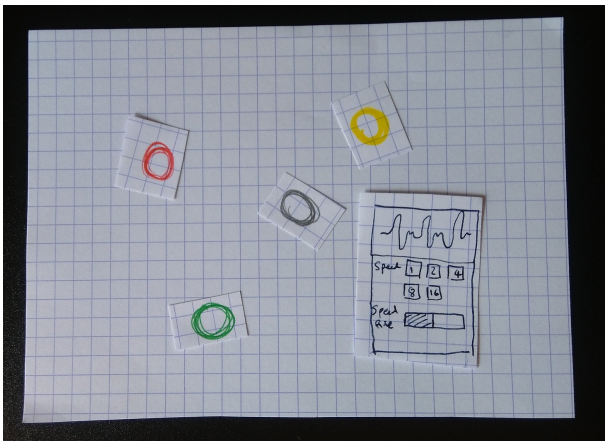

Figure 9.1: Example of paper prototype showing nodes and menu

Formal Testing

I recruited four expert musicians, three professionals and one expert amateur, to help shape the system. I worked with them using think aloud usability studies of Neurythmic, and structured creative tasks, both with their own instruments and Neurythmic, followed by interviews. I transcribed these interviews and studies and conducted thematic analysis on the resulting transcripts following the approach described in [36]. The results of this analysis are discussed in the context of development in the next chapter, and in the final evaluation in chapter 11.

The formal think-aloud testing and explicitation interview described below were carried out at the middle stage of the development described in chapter 10.

Participants

I ran my study with four subjects. To maximise the value of the results, and to focus on the target market for this kind of system, the subjects are all expert musicians - 3 professional and 1 expert-amateur. All users have experience making music with computers, though only two use computers as their primary environment.

All subjects are male, aged between 25 and 40 and living in Bristol. This is an unfortunately narrow demographic, resulting in part from time constraints and the need to recruit subjects from the pool of musicians of my acquaintance. Ideally a broader user group would have been preferred, being more representative of the community of professional musicians.

User 1 Paul Jebanasam is a composer, sound designer for film trailers, label manager and a lecturer in music. Though mainly working with electronic instruments, at the time of his participation in this study his main work in progress was a commission for soloists of the Deutsches Symphonie Orchester.

User 2 Seth Cooke is a drummer, percussionist, composer of electronic music and field recordist working in experimental music. He is the only one of the group who has not made a living in music, though he has had work published by labels in the UK, Europe and the US. He has been interviewed about his work in the highly respected experimental music magazine The Wire.

User 3 Robin Allender is a guitarist, songwriter and touring session musician who has worked with bands like Gravenhurst and *Amelie* soundtrack composer Yann Tiersen. His music is recorded and edited using Digital Audio Workstation software, but he stated that he does not compose directly on electronic instruments. Instead he uses them to augment compositions written on guitar.

User 4 Dominic Lash is a composer, academic and improvising double bassist who works primarily in Jazz, improvised music and modern, minimal, composition. While some of his works incorporate electronic elements and many of his works have been constructed using Digital Audio Workstations, he does not consider himself an electronic musician.

Think-Aloud Study

Think aloud testing was chosen as the main formative evaluation technique to support development. It was chosen as an easy to learn, and easy to explain protocol which did not limit results to themes and topics which I had already anticipated [37]. By asking users to speak freely about anything which came to mind I hoped to be able to identify both use-potential and usability issues with the system, and to avoid guiding this data with my own expectations.

Some of the issues which might caution against think-aloud [38] were not issues in this case. Reaction time and task-times were not factors, and so its distortion by self-narration was not an issue.

To minimise problems with users trying to please me, or avoid offending me,[38] I emphasised at the beginning that this was a prototype, that my work would not be assessed based on their feedback, and that the study aimed at identifying issues, new features and potential future use cases, to help develop an improved version of the tool.

To minimise problems with users self-reporting greater understanding of the system better than they actually had, I asked clarifying questions, particularly in cases where self-description seemed at odds with behaviour.

An immediate approach to think aloud was used, rather than a retrospective interview, in order to identify bugs and minor issues which might otherwise be forgotten. In the final evaluation (ch. 11) I take a retrospective Think Aloud approach.

The specific details of my testing were as follow:

Testing Protocol

Each participant was asked to read a task description (see appendix B) then given basic information about the system and asked to interact with it freely. Testing progressed through three stages, described below, during which features were progressively turned on and information was given.

While interacting with the system, users were invited to speak their thoughts aloud. When users were silent, depending on context I either allowed the user to continue until they spoke aloud unprompted, or else asked non-leading questions about the activity: “what are you thinking now?”, “do you have any questions?”. Occasionally these prompting questions focused on a particular feature: “you have returned to that feature a few times, what do

you think it does?". Apart from this, questions were used to clarify my understanding of a user's statement.

Study Stages

I explained the basic principles of the system and then allowed them to play freely. To focus attention on particular aspects of the system, and accelerate the process of learning (since only a short period was available), the features of the system were introduced progressively, in 3 stages.

1. The session began with a simple preset already loaded for the user(the same network each time).
   - Percussive sounds from the "drum kit" (see section 10) were used
   - The network created a generic, jazz-ish rhythm in 4/4.
   - The connections in the network were mostly feed-forward, though one quite low-weight feedback connection was included.
2. Users could only access the gestural interface - meaning that they could only move nodes (thus changing connection weights), not add them or change their properties. The aim of this stage was to focus attention purely on the gestural interface. Users in informal pilot studies had occasionally become quite preoccupied with changing sounds and pitches, and one pilot user had created a network in which gestural change made little difference. Given the low numbers in the study, I wanted to ensure all users fed back on all parts of the system.
   During this stage I tried to avoid describing how features worked, or explaining too much, as in part I wanted to understand how discoverable the features of the interface were. In one case during this section the user stated that he would find the system more satisfying if he could work with melodic material, and so I loaded a melodic preset.
3. Using the same preset, access to the node-menu was enabled. This allowed users to add and remove nodes, change individual connections, change sounds, and generally access all features of the system (described in ch. 10). I began by answering any unanswered questions from the first part, introduced the basic features in the menu and then, again, encouraged them to play and speak their thoughts aloud.
4. In the final section I reset the system to a beginning state, with only one node on screen, and again invited users to play and speak their thoughts aloud.
   This section aimed to isolate issues around general creativity with the system, and the difficulty of building a network from scratch:
   - How easily could users build a musical network themselves
   - Did the system guide users towards similar results or support diverse creativity
   - What musical qualities did different users pursue, and how successful did they feel they were in this

Expert User Task and Explicitation Interview

In addition to the Think Aloud study above, I carried out an expert-user task with each user, followed by an explicitation interview. This approach was informed by the use of explicitation interviews in a HCI study on tactile experiences [39], and my approach drew greatly on accounts of applications of the technique in [40].

Unfortunately due to time constraints on the project, and the volume of transcription necessitated by expert-user interviews, the results of this process are not included in this thesis. I hope to include this data in a future paper on Neurythmic.

Nonetheless, as will be discussed below, even in the absence of formal analysis of the results, the process was incredibly valuable in developing rapport with the users, training users in self-narration ahead of other evaluations, and understanding the creative perspectives, which would influence users' feedback.

Rationale

In conducting these interviews I aimed to understand the way that participants' instruments participated in their creative activity. This followed from insights from the literature discussed in ch.5, particularly Bertelsen's account of Instrumentness [28].

Bertelsen's account described the way users interacted with digital instruments, and was drawn from interviews with two participants. Analysis in that paper was influential on my development, and I was keen to expand the scope and evidence base of that paper, this time including professional musicians of varying backgrounds.

Explicitation interviews are valued in psychology and business research as a means of accessing unconscious expert behaviour [40], and I hoped to identify some sub-reflective aspects of musicians' interaction with instruments not easily accessed by other approaches.

I also hoped that data from this interview could stand as baseline for my subsequent final evaluation of Neurythmic. Accordingly the form of the final evaluation task closely mirrors that of the expert user task discussed here.

As discussed above, the analysis which would lead to this was not possible within the time-frame of this project. Nonetheless, the process increased my own understanding of participants' processes and built rapport and mutual understanding.

Priming for Think Aloud

I placed the creative task and explicitation study in the same session as, and immediately before, the think-aloud study discussed above. The goal was to prepare and prime users for the self-narration required by that latter process.

The usability researcher Jakob Nielsen suggests that the results of think aloud studies can be improved if the user is first shown a demonstration of what is required - in his example, a video of a user thinking aloud. Specifically Nielsen suggests that this is helpful in getting users talking in the first place, and in guiding the user towards the kinds of things the researcher is keen to learn about [41]. I attempted to address this same issue by allowing the users to rehearse self-narration of creative practice in a situation where I guiding them towards details in which I was interested.

Protocol

task

Participants were asked to imagine they were commissioned by a television producer to create music for a new project. The style and details of the project were to be of the users' own imagining. In this session the user was asked, using an instrument of their choosing, to begin developing material that they could take into a first meeting with the producer.

The task was specified in this way in order to avoid the aimlessness of free play and focus the user on creating a particular result, while ensuring that that particular result fitted the user's own style and musical approach. The full description of this task is available in appendix A

Preparation

Ahead of the task, and in the task description I focused on developing a contract between myself and the participant. I explained clearly how the session would progress. I emphasised that I would be focusing on small details of their behaviour, and that I would be asking them to focus quite intently, and delve quite deeply, into the experience of the task. After each major detail I asked if the user was happy with that approach.

Interview

The task described above was followed by an explicitation interview. Since the explicitation technique has many elements and the results could not be included in this process I will only cover some aspects of the technique used here. An excellent source for details of theory and practical application of the technique can be found in [40] and in [42].

In general terms the technique focuses on eliciting "evocation" states in the interviewee - moments where the experience under investigation is evoked strongly in their mind. The technique thus concentrates on putting the interviewee at ease, and encouraging description of sensory experience at the moment under investigation, in an attempt to recover the adjacent thoughts. In broad terms it applies something like the principle of walking back to a room where you had a thought in order to remember its details again.

The technique is time consuming and takes effort to learn - I found it necessary to run a couple of rehearsal interviews with friends ahead of the studies themselves. It also produces a lot of material to be analysed. As such it is not appropriate for every case. Though this is incidental to the study results themselves, all participants reported finding the experience rewarding - something which might be considered for its value in building trust and motivation where studies stretch over several sessions.

Chapter 10
Development

Overview

In this chapter I discuss the development of the Neurythmic application. This development progressed in two stages. The first stage followed from prototyping and resulted in the first version of Neurythmic. This was then tested with four expert musicians as described in the previous chapter.

The second stage of development resulted in the final Neurythmic application. This second stage improved the basic application based on insights from two main sets of data:

1. Thematic analysis of transcripts of think aloud testing of the first version (see ch. 9)
2. Results from analysis of MNO behaviour, GUIded by 1) (see ch. 7)

Presenting this in two distinct stages split by detailed analysis of testing results proved verbose and repetitive. For readability the account below is instead organised around the description of particular elements of the system in their development from start to end. In each case I give an overview of the first solution, then describe improvements made in the second phase, briefly discussing the user feedback which prompted them.

System Overview

The program divides into two layers - an engine for musical applications of CPGs, and an interface and multimedia layer. Elements of both of these layers are mingled in the discussion in the first part of this chapter but with a focus on novel aspects of design rather than on commonplace aspects of implementation. Fuller details of technical implementation of the engine are presented at the end, the tools used are discussed in ch.8.

Except where explicitly stated otherwise, every design element and implementation discussed in this chapter is my own work. In the code itself the handful of borrowings made from other sources are explicitly flagged with comments.

Metonymic design

The design approach taken was broadly influenced by Bertelsen's account of "instrumentness" in [28] (discussed further in 5). In particular I took a more metonymic than metaphorical approach to specifying interface and function. To unpack that a little, I did not take the very common approach of moulding function or presentation around the image of a familiar instrument or musical concept - as when software delay units imitate the interface of tape-echo hardware, or when software sequencers imitate the look of the Roland 808. Instead I tried to make presentation and interaction follow from materiality.

In broad strokes this means that, for example, the interface is presented as a network, reflecting the underlying system processes. The user is given the means to manipulate that

Figure 10.1: The Roland 808 hardware drum machine - a familiar design metaphor used in digital sequencers

network, with the interface supporting mapping to fundamental, musical functions like timing offset, volume, pitch. Even later when musical constraints, such as quantisation, are added, I provide means to remove or reduce the effect of these.

Following Bertelsen I hoped that this approach would allow users to reason freely about the system outside of metaphors which GUIde intended use, and this in turn would support them in developing their own creative approaches to the tool. I believe the breadth of musical outputs evidenced in my small testing group somewhat vindicates this approach.

At the same time I did not take a purist metonymic approach - there are elements which lean towards metaphor. For example, when I added musical constraints to the system, the interface for some of these borrowed the familiar shape of the step sequencer (see fig. 10.12). This step sequencer metaphor in fact breaks down quite quickly as dots here do not represent notes, placed precisely on the grid, but rather points at which notes are allowed to fire - and these cannot be placed arbitrarily, but are regularly spaced.

In cases like this, the decision was made from a concern for communicability - familiar design tropes come with their own expected affordances and these can be used to increase the immediate explanatory power of an interface. This proved broadly successful in the short testing possible during this thesis, but over the longer term it will be interesting to monitor the degree to which, as we might expect from Bertelsen's account, these design metaphors and their implied affordances place constraints on use and imagination which do not follow from the system's underlying processes.

Interface Overview

In both phases of Neurythmic's development, interaction is split into two modes. There is a draggable network-representation for immediate control of network weights (below: 10), and a detail menu, accessed from the node, which allowed more fine-grained control of node-level details.

The graph representation is illustrated in early form in fig. 10.2 and its development and improvement is discussed in more detail in 10. The latter detail control is shown

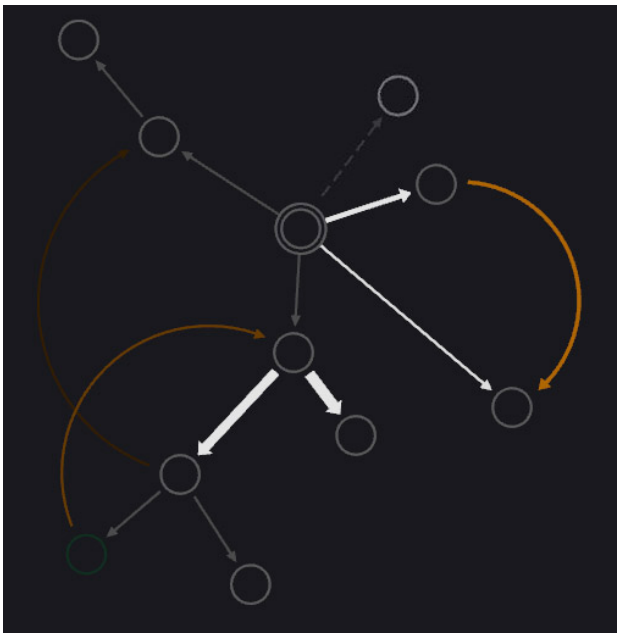

Figure 10.2: The network interface in the first version of Neurythmic, thickness and brightness of connection indicates strength. White connections are "parent-child" connections

in early form in 10.8 and discussed in 10. In both of these sections I discuss individual elements of the design in the context of their development and user feedback.

Broadly this design allowed users to specify a network in detail, its connections, node-rates and output sounds in detail via the detail menu, and then control behaviour of overall network rhythms by the intuitive mode of clicking and dragging nodes on the screen. I hoped this would allow users to keep in mind only what was relevant for a particular interaction mode - hiding distracting low level detail when, in "performance" mode, while keeping it accessible should it be required.

In the first phase of development I did not introduce further structure into the interface. This was done in order to avoid overly-structuring user behaviour, as well as to allow for fast changes to the design. While the simplicity of this design served its purpose well, user testing indicated the need to elaborate on this design.

Network Representation

The problem

In working with prototypes I found it useful to be able to make changes to the behaviour of groups of nodes at once, by changing the weight of multiple connections (see section 12). Since individual control of the numerous network weights does not support this kind of interaction, it was important to find a representation and interaction method which did.

Prototyping had also shown me that the networks were not always predictable in detail, but that this was not necessarily a problem. It remained quite possible to mould rhythms via hands-on interaction and progressive approximation (see sec. 3). In fact in light of this, the unpredictability was often appealing - creating a sense of discovery and a different relationship to the material than if one had specified it more immediately. I was therefore keen to support intuitive, exploratory interaction with the system, and evaluate

user responses to these qualities.

Finally I was keen to find a solution which made it easier to navigate the complexity of the system comprised of so many parameters, whose numeric representations did not map to intuitive musical categories. It was obvious that the interface and network representation should provide clear visual display summarising this detail in an easy-to-digest manner.

Initial Solution

Based on the principle that it is almost always better to borrow an existing, successful, idea than risk a new one of your own, I developed an interface for the network following the pattern of commonly used visualisations for graph structures. The first version of this, evaluated in formal formative testing, can be seen at the start of the chapter in fig. 10.2.

Each oscillator-node in the network is drawn as a circle, with connections drawn as arrows, indicating direction of signal flow. Using a familiar pattern in this way allowed me to rely on users' prior experience of such diagrams, and thus dispense with the explanatory text commonly recommended in discussions of UX design [43].

Nodes in this interface can be clicked-and-dragged. This gives a means of controlling groups of connection weights, since the weights of connections between nodes are scaled by the distance between them. We will see later that individual connection weights could also be controlled via the detail interface, with the detail control setting the base weight, and length-of-connection scaling that value. The latter control gives a gestural, less precise means of controlling the behaviour of groups of nodes.

This weight was displayed on screen via brightness and thickness of the connection. In addition the whole network can be dragged by right clicking on blank space, to allow users easily to organise their workspace.

To indicate activity in the network, the nodes "pulse" as they fire. This is represented by a brightening of the node-colour and a thickening of its line representation. Both of these parameters are driven by an envelope class which is triggered when the node triggers a note. Drawing of line thickness, in both the node and the connections is handled by an open source line-geometry shader.

Finally, connections with zero weight were shown as dotted lines using a shader I wrote myself.

Hierarchy

As discussed in the section 6, I had found it useful when prototyping to impose a hierarchy on the network. I carried this into the first system design, distinGUIshing between "parent-child" connections which established hierarchy (shown by straight white lines), and non-hierarchical connections (shown by curved orange lines)

This hierarchy was used primarily for control of node frequency - a node's frequency was always set as a multiple of its parent. The aim here was to allow easy specification and control of network subgroups (see sec. 12). Changing parent frequency would change frequencies of all child nodes proportionally.

Supporting this further, the interface only allowed the adding of new nodes as children of existing nodes.

We will see later that this approach was not successful and the network hierarchy was removed in the final system.

The Final Network Interface

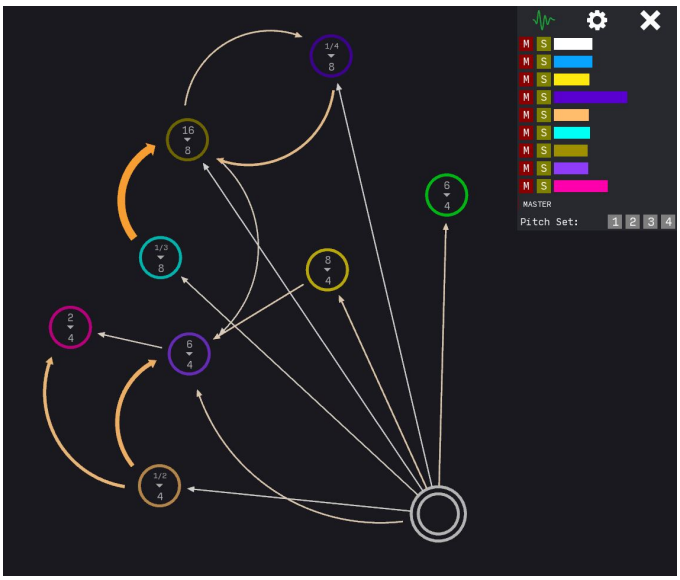

Figure 10.3: Screenshot of the final design of the Neurythmic interface

In response to user feedback during formative evaluation, the interface described above was augmented in the final system, adding two further elements. This final system breaks down into

1. Draggable nodes
2. Individual connection changes
3. Audio mixer

This interface thus gathers together the parts of the system which can be explored with little or no understanding of function, providing the basic tools to begin to learn how the system works. This is the interface presented to the user when they open the application. It is designed such that a beginner user could play a pre-built tutorial network using no other resources than are laid out in it.

The interface is to a large degree flat - without hierarchy between sections, so that all parts are accessible at all times. It defines a kind of performance interface, and the design of individual parts focuses on minimising mousing distance and number of clicks. Almost all functions are available via a single click, the exception being tweaking of individual node connections, which is accessed by a single-level menu, opened directly by right clicking a connection.

Draggable Nodes Interface

This aspect of the interface remains almost unchanged from the initial prototype. The basic mechanic was simple and intuitive and no issues with it were identified in the think-aloud studies. Only details of the underlying behaviour were tweaked. In this version weight of connections is scaled according to frequency ratio of the two nodes connected to ensure consistent behaviour (see sec 7 for background and details of how the curve for this mapping was generated).

Figure 10.4: left: "Which input is 1?!" Screenshot from previous prototype showing ambiGUIty of the input interface. Unclear which input is which. Several clicks to find out.
right: in final version, connection menu opened by clicking directly on connection: 1 click, no uncertainty

Individual Connection Changes Interface

In both versions of the system, the strength of connections can be set individually, or at group level. Users set connection weight on individual connections and this is scaled by the on-screen distance between the connected nodes. This two-part control allows fine tuning of the network and also supports the ability to organise sound outputs by spatially clustered sub-groups.

In the initial build, individual connections were controlled via the detail menu. This showed the list of inputs to the node. When one input was selected, the weight-scaling and time-offset of the input signal could be changed via sliders (see fig 10.4. Users found this interface somewhat clumsy, having to click through several the options to find the right connection.

To address this I moved access to controls for connections out of the detail menu and into the main interface - accessed by right clicking on the connection to be changed. Controls are then displayed beside the pointer, allowing modification of the connection's weight and time offset, or deletion of the connection. (fig10.4).

This change required changes to the interface code to store details of on-screen placement of connections. It also required that I implement simple ray-casting algorithms to identify when a click or mouse-hover was within the representation's outline.

Audio Mixer Interface

The addition of an audio interface was a common request in the think-aloud studies. Users explained the request both in terms of improving ability to perform with the system, and in terms of separating mixing from the other menu functions, to avoid potential error.

The mixer is intentionally simple and follows conventional mixer standards to leverage users' awareness of the common affordances of such interfaces. It consists of a channel for each node, each consisting of conventional representations of mixer mute and solo buttons and volume slider. The slider's colour matches that of the node whose output it controls to ease identification. Mute and solo functions address the request from users for an easier means of silencing groups of nodes.

This GUI element was built by myself from basic openFrameworks drawing elements and mouse-action callbacks. I developed a mixer class containing a series of Channel classes,

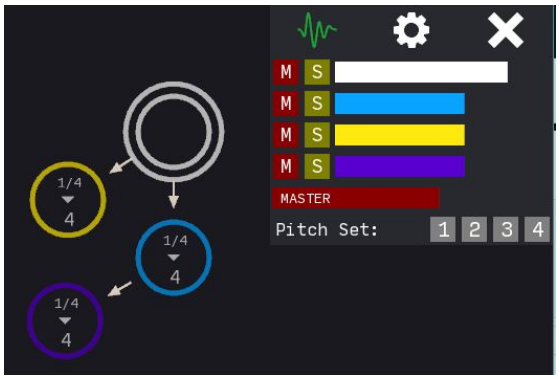

Figure 10.5: the mixer interface, and the nodes whose audio outputs it mixes

themselves making use of Button and Slider classes. When nodes are added on-screen they are added to the mixer, when they are removed from the network, they are removed from the mixer.

Network Hierarchy and Representation of Connections

Think aloud testing of the first version of Neurythmic raised three main issues around the representation of the network.

Visibility The first version showed the strength of a connection by the thickness of the connection's line, and by its brightness. An unintended side-effect of this was that when connections were weak (thin and dim), users sometimes had trouble seeing them.

Legibility Users also seemed to have trouble reading the meaning of the lines. Specifically the change in line width was too subtle, and the visible range too narrow to make meaningful distinctions.

Clarity The most fundamental of these three issues. The distinction between hierarchical parent-child connections and non-hierarchical connections was confusing to users (see 6. The first prototype used the parent-child structure to allow users to set node frequency of a child as a multiple of the frequency of parent node. This was designed to allow easy control of whole branches of the network (changing a parent frequency changed the frequency of all nodes beneath it proportionally). However, users did not seem to take advantage of this to organise their networks, and found it distracting to multiply frequencies in their head - particularly two or three levels down the tree.

In my redesign I have removed this confusing distinction between connection types, to simplify the UI. All connections are now represented using the same colour scheme for ease of reading, and all frequency multiples refer back to a single root tempo (frequency of the root node), as in a conventional sequencer. This is intended to make the behaviour resulting from a frequency change far more straightforward to predict.

While removing this distinction, I have retained the requirement to create new nodes as children of existing nodes, though it is no longer strictly necessary. This was retained purely due to time constraints - a change here would require too much redesign and refactoring to fit into the development period I had left.

Representation of Connections

In addition to removing the hierarchical distinction described above, I also revisited representation of connections, since the original design principles had been invalidated by the

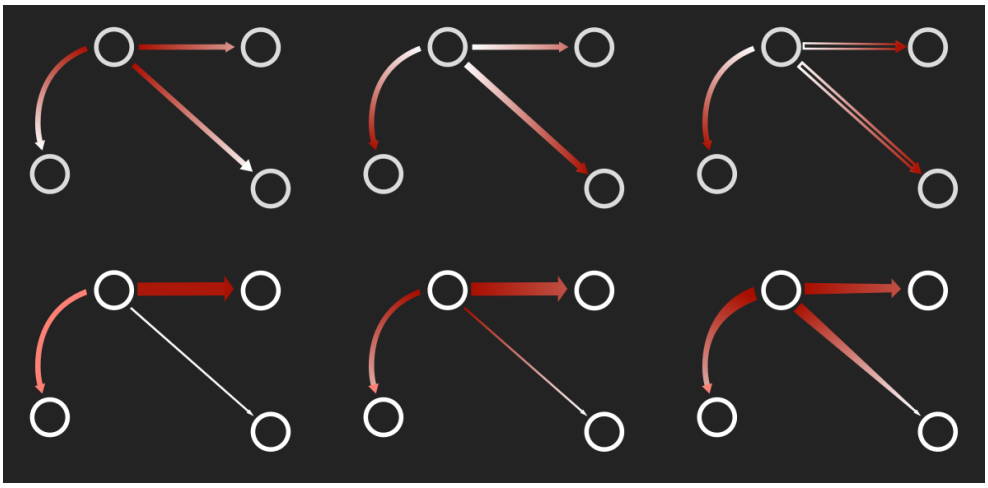

Figure 10.6: Sketches of connection representation schemes tested with users. When these were developed I intended to retain the distinction between parent-child and "normal" connection, and some designs aim to emphasise this distinction. The chosen scheme was the bottom left.

feedback discussed above.

New designs were quickly brainstormed and sketched, and then and informally tested with available colleagues from the course. These sketches are shown in figure 10.6. All designs moved away from representing connection strength with a bright-to-dim colour scheme, using a coloured-to-white colour scheme instead. This ensured the visibility of all connections.

I explained to the testees that the sketches represented networks, and that the lines represented strength of connection. I then asked them first to describe the situation in each image. The most consistent interpretation of meaning was given to the design in the bottom left: all those questioned identified the thick red line as the strongest, and the thin white line as the weakest.

Figure 10.7 shows the final scheme used in the application. Variation in line width is larger than in the first prototype. This is to allow easier estimation of connection weight at a glance, an issue identified in formative testing. As discussed above, both weight scaling (re-christened "strength") and distance between nodes affects connection weight. The figures show weights differing due to both of these factors.

Node Detail Interface

As discussed above, the application interface retained its division into two parts all the way through development. I intended that users should be able to build a network using the detail available by precise slider GUIs, but then move to a largely freer, less fiddly mode of interaction once the network was built, using the interface described in the section above.

So while the network representation informs the user about the overall state of the network and allows control of connection weights in groups, the node menu allows the control of detail. Here I included controls for the other parameters I found useful in working with the prototypes, but which I felt I returned to less often.

In both phases of development a "node detail menu" was accessed by right clicking on a node. The GUI elements in that menu controlled behaviour of the node selected.

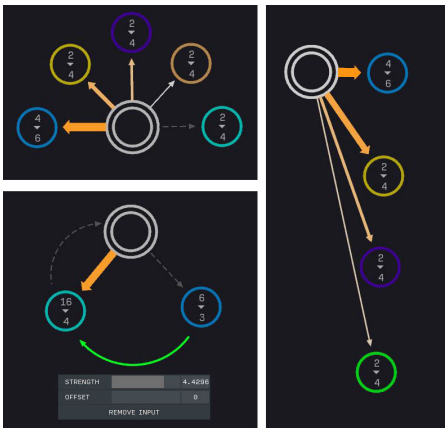

Figure 10.7: Top left: connections at same distance, but with different weight scaling, right: connections with the same weight scaling but at different distances, bottom left: the types of connection - straight parent-child connections and curved "normal" connections. Broken lines indicate zero signal. Connection controls, displayed at the bottom, are for the highlighted connection

Figure 10.8: View of the first design of the node menu, accessed by right-clicking a node

Initial Solution

For simplicity of development, the first version of Neurythmic used a single screen for the detail menu, allowing for the following actions. See fig 10.8 for illustration. Most of these elements carry over to the final version, though often changed.

- **Control of node frequency ("Rate")**
  Two controls allow course and fine tuning. Integer beat divisions, commonly used in drum machines and sequencers, are available at the top, and these can be fine tuned, allowing non-integer divisions (see 12). As discussed above (6), in the first design frequency is set as a multiple of the frequency of the parent node.
  These controls are contextual. When the menu is opened from the root node, this control shows a value for Beats Per Minute (BPM) instead, and sets the root node tempo. This is nominally the overall tempo of the system. This is true in both early and final versions of Neurythmic.
- **Control of weight and signal offset of individual input signals**
  Individual inputs are selected by clicking one of the buttons in the "select input" control. This input is then highlighted on the network interface, and weight scaling ("strength") and timing offset can be set for the connection. We will see later that this element was found confusing and so changed in the final system.
- **Adding a new node**
  As discussed above (6), new nodes can only be added as child of the current node, in order to clarify network interaction and support productive patterns of interaction.
- **Deleting the current node**
  This control is contextual - to maintain network hierarchy (6) only leaf nodes can be deleted, and this control only appears for those nodes.
- **Specification of the parameters of the output sound** This allows control of pitch,

timbre and volume. A piano GUI was developed to provide a familiar interface for pitch control. 6 synthesizer timbre presets are provided. For further discussion of the sound output see 10

Display of the Waveform

In addition to the controls discussed above, the signal output of the node is displayed at the top of the menu. This may be a largely aesthetic feature: I have only occasionally been aware of its utility in diagnosing the behaviour. Nonetheless before this was added, more than one user with whom I informally tested the system asked if it would be possible to see the node signals.

Implementation of Menus

In this section and in the remainder of the project, menu elements (except where explicitly indicated otherwise), were built using the ofxDatGUI library - a simple modular menu extension to the openFrameworks framework. This allowed for fast development, addition and removal of menu items, at the cost of some quite acceptable limits on visual customisation.

I adapted and extended this library in some cases to fit my purposes better, adding new GUI elements not available in the core library, and altering other classes to add functionality I required. This will always be noted in the thesis text. During this first stage of development I extended the framework by developing the piano GUI element, and altered the button-matrix to support user-specified text labels.

Musical Output

In a commercial tool of this sort I would expect to see either the ability to control external sound sources (c.f. Bertelsen's discussion of the modularity of musical tools in [28]), or else a versatile sound engine allowing considerable user customisation.

In this case, design considerations focused on testing of usage scenarios, not on wider distribution. While the ability to connect the system to users' own sound sources would make the test situation closer to real world usage, and was a regularly requested feature, I felt that for the purposes of this thesis more could be learned by focusing attention away from sound-generation systems and towards rhythmic output. As such I purposefully allowed users access to only a limited set of quite generic sounds.

Wanting to focus on rhythmic interaction, I used percussive sounds, with short attack and decay envelopes to ensure clarity of the patterns generated. The synth voices I designed aimed to cover the basic elements of a traditional drumkit (kick, snare, brush, hi-hat) and simple melodic sounds (marimba, bell). These were designed to be adequate but generic enough to avoid being ear-catching, trying to focusing attention on patterns and away from sounds.

Synthesizer Implementation

I developed a Simple Frequency Modulation synthesis engine ([3], p. 227) for the application using the C++ synthesis library Tonic. Frequency modulation synthesis was chosen for its balance of versatility and computational cheapness ([44], p.457). I developed a basic interface for this to allow me to design a set of presets, but only allowed users access to the

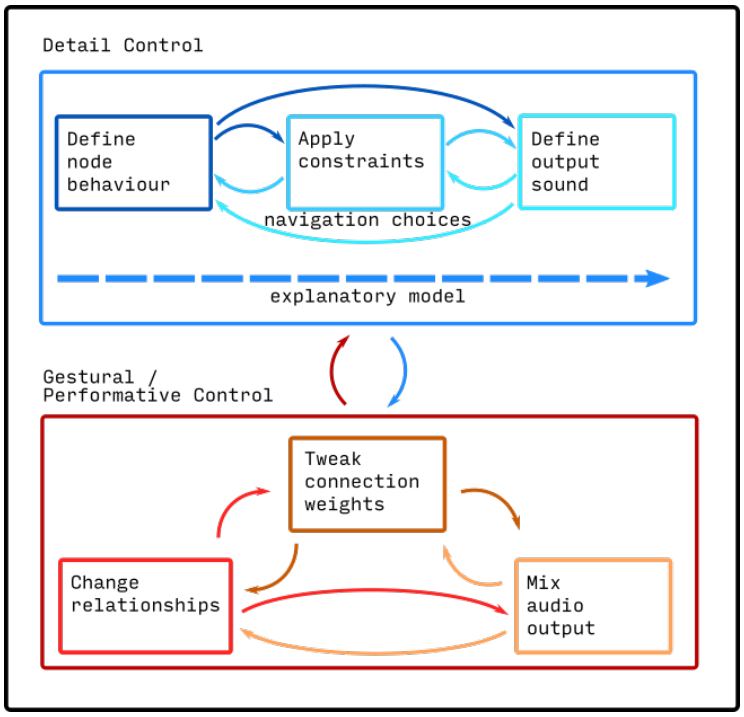

Figure 10.9: Organisation of the interface showing both main divisions, navigation options(solid arrows) and the three part, sequential, explanatory model used in the detail menu

presets. This was to avoid distracting users by allowing them to play with the synthesis engine.

Presets and Interface Modularity

To enable me to repeat test scenarios with subjects, I implemented a preset system capable of storing momentary system state to an XML file, and recalling it.

Also to support testing I built the interface to use a global settings class, which loaded its values from another XML file, and which could be reloaded during operation. As well as defining most visual parameters of the application, allowing me to tweak them interactively, this allowed me to turn certain interface features and menu-items on and off, supporting support testing scenarios (see ch.9 and ch.11).

Improvements to the Detail Interface

Explanatory Model

When incorporating the identified improvements from the think aloud study in the second phase of development, I redesigned the node-detail interface to provide an implicit conceptual model for interaction with the system, to improve usability.

The goal here, following Don Norman's observations about conceptual models in *The Design of Everyday Things* [45, p. 25], is to provide a mental shortcut for use. The model aims to wrap more complex processes in an approximation which supports intuitive reasoning about actions and their results. I developed such a model by dividing the UI for the system into functional groups, adjusting these into a compromise between an organisation which grouped controls likely to be used together, and an organisation which followed

Figure 10.10: View of the detail menu, showing the three sub-menus, comprising a three part model of system behaviour. Here the menu is open on the Node Behaviour tab

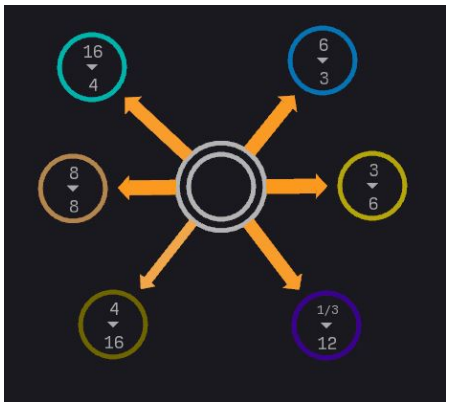

Figure 10.11: Information display in the gestural interface. Frequency at the top, "flows into" the number of beats per bar the node output is then constrained to by the quantiser

actual order of operation in the system (where that ordering had some effect on behaviour).

The final version of the system presents a three stage model for how the system works. It is represented on screen by the ordering of the three sub-menus which comprise the detail interface. This visual ordering aims to imply a process without forcing an order on interaction: While any menu can be accessed at any time, the movement from left to right implies a sequence of operation which broadly follows actual system function. This representation of the model is on screen whenever the user is interacting with the system details, so that it does not need to be kept consciously in mind. (see Don Norman's "Information in the World" principle [45, p. 111].

The detail menu is opened by right clicking a node, implying to the user from the start that the clicked node is the one under control. This is further emphasised by the fact that the node under control is highlighted in a bright green used consistently throughout the application to indicate selection.

The menu itself is organised into three sub-menus, and when one tab is selected, the other options are occluded to help users focus on one task or area of operation at a time. The ordering of these three tabs aims to present a conceptual model about the functioning of the system and interaction with the node.

The three menus are as follow:

Node Behaviour Starting with the first tab, the user defines the node's underlying behaviour. To emphasise this, the signal output of the node is shown.

Constraint The second tab allows the user to apply constraints and transformations to this basic behaviour. Controls here determine the degree to which rhythmic output locks to a grid, and the shape of that grid (see section 10 for more detail)

Sound Output The third tab allows the user to define the sonic output of the node - its pitch, timbre and volume.

Again following Bertelsen's recommendation of metonymic design in instrument interfaces [28], this structure maps fairly closely to the order of operation in the underlying system, with only small deviations to improve explanatory power (signal delay, for example, is manipulated in the constraint menu, but depending on settings, may be affecting the node's underlying behaviour).

Figure 10.12: Sketches for the logo designs. The logos highlighted in red are the ones used in practice.

Detail Information Displayed in Gestural Interface

To reinforce the conceptual model described above, details from it are repeated in the network interface. Here, as shown in figure 10.11 the frequency of a node (from the first menu) is represented by a number printed at the top of the node, followed by an arrow implying input, or direction of flow. Beneath that is printed a number indicating the number of beats to a bar to which the node is constrained (from the second, constraint menu).

This information display is on screen at all times, serving the dual purposes of reinforcing the order of operations in the conceptual model, and providing key pieces of information necessary to predict the effect of changing a node connection. Users would likely struggle to keep these values in mind without information on screen.

I found that these numbers together were key to understanding the behaviour of a node. They determine behaviour completely when a node has no signal inputs, and even when input connections are present they still give a strong clue to present behaviour, and the bounds within which it may be changed. We will see that users judged the final version of the system, in which these numbers were present, easier to use, though none commented directly on their presence.

Icons

Rather than using text to indicate these sections, I opted to use icons. This decision goes somewhat against recommended practice and usability research (e.g. [43]) which indicates that outside of highly familiar icons which have acquired standard meanings in culture, icon meaning is not communicable without text.

Nonetheless I reasoned that within music software certain icons - such as waveforms and notes for sound, grids and dots for quantised-time interfaces - were established and could be reused here. Further, where Neurythmic departed from familiar concepts, the application interface had itself already established meaning for certain visual elements, with for example nodes (and no other elements) represented by circles. I felt these visual elements could easily be reused, creating a consistent symbolic system within the app.

Again, I quickly made sketches of options that occurred to me, and tested these informally with colleagues who had used Neurythmic. The icons tested in this way are shown in figure 10.12. I showed users these icons and asked what they represented within the app. Those chosen were the ones which received responses which consistently approximated intended function.

The icons were drawn in Inkscape, saved as SVG and loaded into my application using

Figure 10.13: View of the node behaviour menu, open on the root node

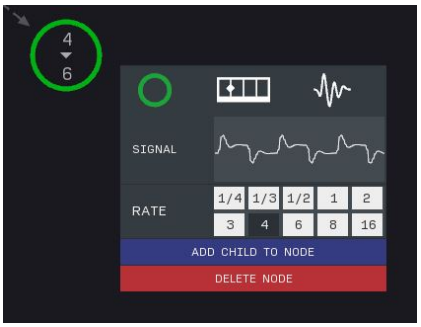

Figure 10.14: View of the node behaviour menu, open on a child node

Figure 10.15: View of the constraint menu, open on the root node

Figure 10.16: View of the constraint menu, open on a child node

the ofxSVG library, whereupon they were saved in Vertex Buffer Objects for fast drawing by GPU. The tab GUI, used for selecting between the menus, was developed myself out of basic openFrameworks drawing elements and mouse-callbacks.

Node Behaviour Menu

This menu is the simplest of the three. It contains controls which affect the behaviour of the node selected. Specifically it allows for control of node frequency, deletion of the node, and adding of children to the node. It also displays the signal output of the node. All these features were found in the first prototype discussed in sec 10.

The only change to function described there concerns the frequency control. In the previous prototype both discrete values and fine control were available, allowing the specification of frequency as irrational multiples of the root beat. While I found this useful in prototyping (see 12), it proved somewhat confusing in user testing, placing emphasis on a very unfamiliar aspect of the underlying mechanic. More than one user needed to ask for explanation here.

All components in this menu are built from the ofxDatGUI menu library.

Constraint Menu

This menu combines controls for the quantiser (see sec. 10), with control of node-output delay. Node delay was reintroduced in the final version of the system, having been excluded from the first version due to user response in informal pilot testing (see 6). I felt that in the context of the explanatory model discussed above, and incorporated into the wider

Figure 10.17: View of the sound output menu

constraint control, this control, which I had myself found to be very useful became much easier to understand.

If the context of later discussion of the quantiser the **grid** and **resolution** controls are quite self explanatory and I will not go into further detail on them here. The **Freedom (Global)** slider controls the strength of quantisation. This breaks with the pattern of interaction in the detail menu since it controls a property of the whole system, not of the node selected. It was included here since I could not find a better place for it at time of development, but it struck me as potentially confusing. We will see in the final evaluation that users agreed.

At the bottom of the menu is a GUI element which displays a representation of the quantiser's effect. The visualisation shows the number of time divisions in a bar chosen, and the points on this timeline towards which notes will be constrained. The GUI also allows the user to set rhythmic offset (shown by a translucent orange overlay) by clicking on the grid. This simultaneously sets the time delay of the node's signal output so that behaviour retains consistency when the strength of the quantiser is changed. This element borrows the time grid representation common in step-sequencer interfaces, and we will see in the evaluation that this led one user to expect it to function somewhat like a step sequencer.

The menu adapts to context - if a root node is selected, the quantiser options are not relevant, since the root node defines bar start (see discussion of the quantiser for details). As such these elements are not shown.

The resolution selector is a new GUI element I added to the ofxDatGUI library. The grid GUI is a class developed myself using basic openFrameworks drawing functions and mouse-callbacks.

Sound Output Menu

This sub-menu collects together the controls which define the output sound for the node. Since I found no usability issues in this area, the controls are taken directly from the previous prototype, unchanged (see sec. 10)

Time Quantisation

The addition of a musical time-quantiser to the system is arguably the single most dramatic change made in the second phase of development. My instinct after user testing was that adding quantisation to the system seemed likely to be the biggest single step that I could take towards fulfilling the system's potential for real-world use by musicians.

Background and Design Considerations

Three of the participants in the think-aloud study were able to create music they were happy with and which they described positively. Users also specifically commented positively on the freedom from quantisation and generation of rhythmic nuance. However, one common negative comment was that it was difficult to move parts of the pattern in time. One user in particular struggled to feel in control and was not pleased with the results he was able to achieve, saying he felt it would be hard to make something "beautiful" with the system.

Musical quantisation aligns notes to a musical time grid, and I felt that something like this could be used to reduce some of the rhythmic unpredictability of the system when desired, and allow users to work more freely. It would also be a good starting point for helping users move notes more precisely in time.

However, was also aware that adding quantisation risked undermining some novel features of the system (see ch.4 & 2). After all one of the main goals had been to free rhythm creation from coarse time grids. Navigating this problem became one of the main priorities of this phase of the project.

Solution: a Variable Quantiser

My solution, described below was to develop a real-time quantiser system geared to these concerned. This system allows for the use of multiple, simultaneous, user-defined, time-grids, and allows for the degree of constraint of notes to grid to be varied continuously. At one end the system will only move generated notes to precise musical grid locations, at the other the quantiser has no effect. In between these extremes the quantiser pushes notes some distance towards the musical grid locations, depending on the setting.

This solution allows rhythmically free and more-or-less constrained voices to coexist in any manner the user requires. In more straightforward scenarios, it allows the user to begin with a conventional constrained rhythm and dial in a little "swing" and "drift" from the underlying network behaviour by lowering the quantiser constraint level.

As discussed above 10 this quantiser was combined in the interface with note offset control to provide a single easy-to-understand interface for directly moving and constraining node outputs in time.

Process Design

Fundamentally, a rhythm quantiser divides time into discrete units, and a system which moves notes onto, or towards, those unit start-points according to some set of rules. Fleshing out the details of this into a working system involves making a trade-off between the desire to retain the character of the underlying source material, and the desire to produce a regular, predictable result. Many approaches to this are possible from the very simple to approaches using Bayesian statistics and AI [46]. The approach below is relatively simple and quick to implement and gives good results. It was implemented in the Engine portion of the system code as a set of C++ classes (those beginning "Quantise...").

Latency

Perhaps the first, most basic decision for a real-time quantiser, is whether or not to introduce a delay into the rhythm generation system. Introducing a delay allows events to

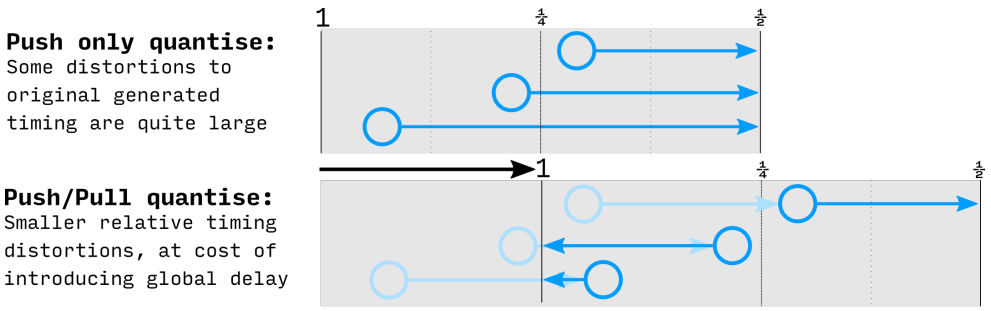

Figure 10.18

be moved both forwards and backwards in time to conform to the grid. If no delay is introduced we improve the system's latency, and considerably simplify design. However, this comes at a cost: we can only move events forwards in time from their original coordinate, and this results in greater deformations of rhythm and less responsiveness to the underlying material. See fig 10.18 below

As discussed above, I wanted to continually vary the degree of quantisation. I also wanted to be able to use quantised and unquantised voices together, allowing expressivity and rhythmic clarity to coexist. These demands prioritise the reduction of relative deformation over the introduction of latency, and so I opted to introduce a delay. As illustrated in fig 10.18 this delay must be at least half as long as the largest division of time we wish to enforce. To allow use of quantised and unquantised notes alongside one another, this delay was introduced to all notes, not only those quantised. No participants noted the introduction of this latency in the final evaluation.

Synchronising Bars

In Neurythmic, the root node of the system is used as a central reference point for tempo. Often this tempo will be consistent, and in this case quantisation is straightforward. However, it is possible for feedback signal into the root node to vary tempo from cycle to cycle. I decided to address this in the most straightforward manner - simply resetting the quantise grid to the firing of the root node. This defines the firing of the root node as the first beat of the bar. The root node is thus not itself quantised and instead drives the quantiser.

When the cycle of the root node is altered by feedback connections this approach creates slight augmentations or truncations of bar length, while the grid units are not changed. Any scheme here is artificial and the choice somewhat aesthetic - a quick test of the concept in MAX/MSP proved the results to work quite well, and we will see that users liked the musical outputs of the system. The exploration of different quantisation schemes stands as a route for future development.

Defining Grids

With bar lines defined by the activity of the root node the system follows the pattern of a conventional, but flexible, rhythm quantiser. Controls are provided for the user to define the number of grid-lines in a bar. Multiples of 3 give a waltz feel, multiples of 2 or 4 give a common time feel. The system supports any number of grid-lines, but in this version of Neurythmic I made available resolutions of 24 and 32 grid divisions for 3/4 and 4/4 time respectively.

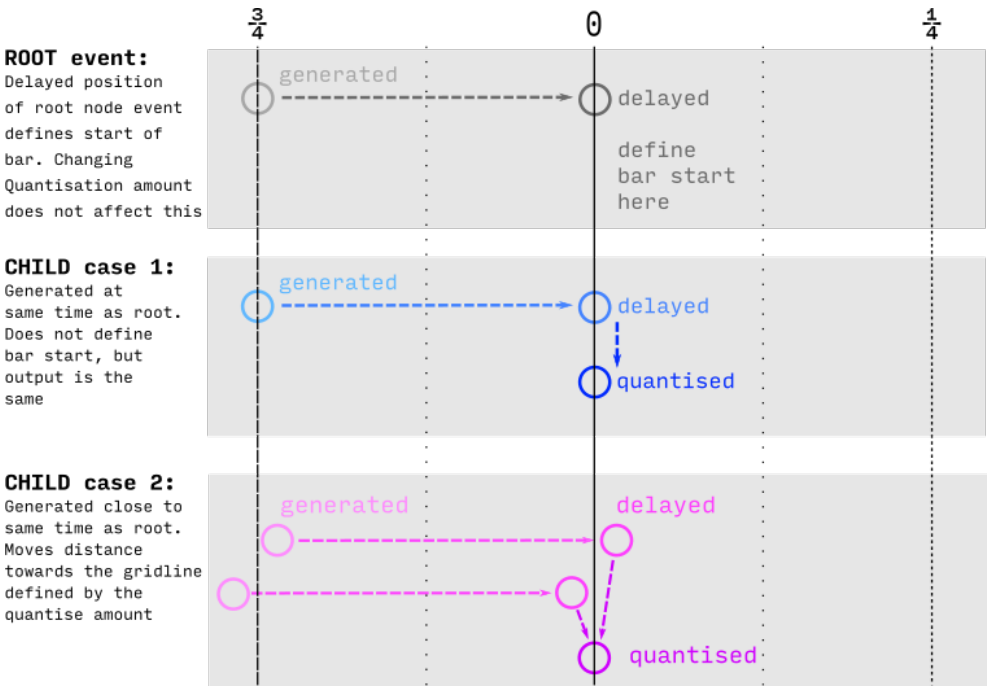

Figure 10.19: Quantiser process. Quantising to 1/2 notes.

After this, the user may choose the note-value to quantise to. This is specified through the interface as "resolution" and describes the number of beats per bar. So, for the 32-grid a resolution of 4 results in 4 beats per bar, spaced 8 grid-lines apart. For a 24-grid a resolution of 6 results in 6 beats per bar, spaced 4 grid-lines apart.

Different grids may be specified per voice, making this a quite precise tool for coordinating overall system behaviour.

Defining Position

To address the desire for more direct control of note placement, while , I added an "offset" option. This offsets the pattern of notes defined by the Grid and Resolution controls by a number of grid units defined by the user. While the user is still not directly placing notes on the grid (actual location will be defined by the network's behaviour) they are now able to constrain, and shift the set of possible note placements to their choosing.

To maintain consistency between quantised and non quantised scenarios, and allow continuous transformation from one to the other, I also added a delay to the node output signal prior to quantisation, but matched to the offset.

Mechanism of Variable Quantisation

With the grid, and valid note placements defined as described above, the quantiser then simply moves the note towards the closest valid note. This process is illustrated in fig 10.19 and described below.

As discussed earlier, this process begins by introducing a short time delay (d) to all notes. So a note is generated by the network at time t_n and is delayed to time $t_n + d$ before being quantised. This delay means that notes can be moved either forward or backwards in time from this delayed point, and the time deformation is thus minimised. (fig 10.18)

From here, variable quantisation is implemented by controlling the amount the note is

moved from $t_n + d$ towards its closest valid grid location g. If quantisation amount is q - a number between 0 and 1, then the note is moved in time a distance of $q(g - t_n + d)$. At $q = 1$ notes always align to grid locations, at $q = 0$ rhythms is unaffected by the quantiser, and in-between the effect is a compromise between the grid and network behaviour.

Engine

I was aware during development that my interface might change considerably in later stages of development, while basic network function would be fixed. I also had ideas for a range of other applications using the underlying CPG process. With both of these considerations in mind I divided the implementation of my evaluation prototype into two parts: the interface and output layer, described above, and the CPG engine which that layer controls, visualises and sonifies.

I developed the engine as a dynamic library, in C++ using the standard libraries. At the core of this I reused the the implementation of Matsuoka's Neural Oscillator (MNO) I developed for MAX/MSP (see 6). As well as simulating the MNO, the engine manages network connections, and provides a simplified interface for control of oscillators and network.

The engine supports the control of oscillators not only using the equation parameters, but also by frequency, using the result and calibration procedure I outline in subsection 6. The engine also allows alignment child node phase to parent, either once per parent cycle, or as a one-off event, using the technique described in 6. Only one-off alignment is used in the systems developed in this paper (once when each child node is spawned, to give predictable and musical behaviour).

This engine runs in openFramework's audio thread to ensure timing stability. Calculations are called once per audio buffer, giving a timing resolution of ˜4ms for a buffer length 192, assuming CD quality (44100hz) sample rate - more than adequate given that studies show that listeners do not readily perceive rhythmic deviations in musical material of less than 10ms [47].

To avoid stalling the audio thread, and to avoid inconsistent calculations, the engine queues incoming parameter change requests from the GUI thread, and actions them together, once per audio buffer, ahead of network calculation and audio output generation. Similarly output events from the nodes are queued for retrieval and action by the GUI thread.

Part IV

Outcomes

Chapter 11
Evaluation

Add Bugs & scare up more criticism?

Overview

This chapter describes final evaluation process for the Neurythmic system, both the process and the results. In overview, three expert musicians, who had been involved in formative evaluation earlier in the project, carry out a specified creative task using Neurythmic. They are then interviewed about the experience, and I analyse the results ofthat interview using the thematic analysis technique.

As in the previous stages, testing was carried out in the participants' own homes. Three musicians are involved at this stage because, unfortunately, participant 3 of the original group had to drop out due to commitments during the final weeks of my thesis. Details of the original group can be found in sec. 9

The protocol for final testing of Neurythmic was designed to mirror the creative task and explicitation interview carried out with these same participants using their own instruments, and described in 9, and I intended to compare results of the two. Unfortunately the volume of transcription (many hours of interview time) generated proved too much for the time period of the project. As such the transcription and thematic analysis of the first own-instrument creative task has not been carried out at time of writing.

The analysis below however serves well as a self-contained evaluation of the system. It reveals successes, musical context and areas for improvement. Further, the earlier task was not wasted - it proved invaluable in developing rapport with the musicians and understanding their requirements (see discussion in 9).

Testing Protocol

Testing was split into four stages

Introductory demonstration and explanation A brief minute session in which I loaded the program, described its general principles and then gave a demo of its features.

10 minute free play session The participants were then allowed free play on the system and ask question about anything that was not clear.

Task The participants were asked to read the task description, ask any questions that arose and then perform the task for as long as it took them - with a suggested time of 15 to 30 minutes.

The task asked participants to imagine they were commissioned by a television producer to create music for a new project. The style and details of the project were to be of the participants' own imagining. In this session the participant was asked to

begin developing material that they could take into a first meeting with the producer. The task was designed to guide participants towards focused creative use of the system rather than just exploration. The aim here was to focus participants' attention on the system's usefulness for focused intentional creation in the participant's own preferred mode, and not only on free play with a novel system.

The full task description presented to participants is available in appendix C

Interview The participants were interviewed about the task in a guided retrospective Think Aloud interview, and the interviews recorded. Questioning began focused on the task outcome - did they feel they had achieved the goal set. This prompted quite open responses containing reflection on the focused use of the system which I would often not have anticipated.

I supported these reflective responses using open questions, by paraphrasing participants' responses back to them, or asking for clarification.

I also used a set of scripted questions covering particular evaluation topics, e.g. "Would you use the system in your own practice", "How would you describe the musical output of the system". These were used if the topics had not already been covered by the participants. I preferred to avoid guiding responses too early, to avoid narrowing down discussion away from aspects of the system the participants might have found significant.

Qualitative analysis

As with the formative studies, the interviews were transcribed, and the transcriptions analysed using the thematic analysis technique described in [36]. This involved reviewing the transcripts in detail and finding themes and subthemes which connected comments across the dataset interviews. In this case some of these themes were driven by questions I wished to ask ahead of the evaluation, and which I had ensured were represented in the discussion. Other themes arose more naturally from participants' responses.

Results

Success in achieving self-specified musical task

All participants reported finding material they were satisfied with, but their levels of satisfaction varied, as did their approaches. Two out of the three reported being able to work towards an intended goal which fitted their personal musical style. **Participant 1** in particular stated that he had found a musical idea during the free play session, related to something in one of his own creative projects, and decided to pursue that in the main task. He reported achieving this without trouble. "that's what I wanted to do. And it felt very achievable".

Participant 2 took a more exploratory approach. He reported abandoning one idea which he found impossible to achieve due to the sounds available, but ultimately generated music which he was satisfied with and which he felt reflected his musical style.

Participant 4 felt he was less successful. He also abandoned one idea due to the unsuitability of the sounds available and ended up pursuing an idea which he had stumbled upon, but had had "no desire at all to make". He joked a few times about not being sure that his work here would ever be used on TV.

Musical Output

Despite being restricted to a limited sound palette, participants used the system to pursue quite different sonic ends. These ranged from drone-like approaches to more "fluid" takes on conventional drum-sequencing tropes. Two of the three users reported being very pleased with the results. All users commented positively on the unquantised nature of the rhythms, and despite the addition of the quantiser system, two of the three users used the system with this either turned off, or turned very low.

Participant 1 in particular used the system in a way I had not expected: as what he called a "surface generator", using a high tempo and structures which he controlled to move between coherence and de-coherence, "very ordered structure" with clear rhythm and more chaotic patterns. He compared this to the way he uses granular synthesis for his sound design work, to specify a "slab" of sound with particular properties and then articulate it.

Participant 2 used the system much more as I had expected it to be used, generating and controlling shifting poly-rhythmic patterns. The participant returned to broadly the same territory and descriptive reference points when using both the first prototype and the final system.

He stated that he was seeking "groove", and in the first session he compared the experience favourably to sequencing rhythm in a DAW, which by comparison he criticised as feeling "like you're very much punching a clock. ... the visual aspect really detracts from the ability to program something that feels like it's got life and a groove."

In the final evaluation session, he described the effect of the system favourably in terms of "slippages", calling the rhythm "unlocked" and "detailed, slightly unhinged". He compared this to aspects of music by Aphex Twin, Photek, and Fourtet. At the same time, though, he emphasised the originality of the results, saying that specifics of the system, such as its generation of variation and free rhythmic placement "tips it over into something that isn't any of those things".

Participant 4 used the system quite differently in the two sessions. In the first he used the system similarly to participant 1, using high tempos to generate droning effects which he articulated by moving nodes. In the final evaluation he tried a similar approach but felt he was unable to get the effect he wanted, so took a more straightforward "drum machine" approach. In part he explained this as due to the sounds available not being suited to his preferred approach. In fact the sounds were the same in both sessions, but I observed in the second session that he used the system with a slightly higher level of rhythmic constraint than the other participants. This moved the emphasis towards discrete events and worked against his intentions.

Finding it difficult to achieve his intended result, he switched approach and made what he described as "a kind of fake xylophone jazz" suitable for "children's tv" - a simple melodic pattern accompanied by a shuffle rhythm in which the system generated variations over a number of bars.

Understanding of the Entrainment Process

Though entrainment is an unfamiliar effect in music technology, all participants indicated that they felt an intuitive understanding of how to work with the system, though when asked if they felt they understood the mechanic itself, they said they did not.

Participant 1 expressed this most clearly: "you don't look beneath the hood - you just see it as a kind of clock sync [...] when you're working with rhythm it's pretty common

sense almost, in a way. It's almost like you're just saying "hey you, pick up a drum and listen to that dude and start syncing". You almost personify them actually."

This self-reported "understanding", in keeping with the system's high-level, gestural approach, high-level and practical, emphasising understanding of musical effect.

Network Representation

Aside from the entrainment mechanic, the other notably novel feature of the system is the representation and manipulation of a rhythm generation system via a 2D graph.

In the first, formative evaluation session, my testing had focused on discoverability and I began by giving the participants very little explanation of the system (see 9). In this case participants expressed confusion about how the spatial representation worked, wondering if the two axes were significant, and being unsure exactly how proximity worked.

By the second session, however, participants reported finding the network representation intuitive. This change was due no doubt partly to prior experience, but perhaps partly to improvements described in 10 which made behaviour more predictable and so, in principle, learn-able.

Most interestingly all participants began to use the spatial representation of the screen to group their system into distinct sub-ensembles - something I had found useful in my own use of the system. Both participants 2 and 4 stated that they found this very useful, giving them an on-screen map of musical qualities. Both participants said they felt this just followed naturally from the system's underlying mechanic, rather than considering it their own creative decision.

Gestural, High Level Control of Rhythm

The control of rhythmic structure at a high level, by gesture, is relatively uncommon and all participants commented on the unfamiliarity of working in this way. Traditional rhythm sequencers work by direct specification of notes. This may have been a factor in why, two participants (1 and 4) took a more timbral / "surface" control approach in one or both evaluation sessions, though one of these was working with transformations between discrete rhythms and timbral surfaces.

Nonetheless, participant 2 reported satisfaction in both sessions when using the system as a tool for conventional rhythm generation. This suggests there is promise in taking a gestural, high-level approach to the specification and control of rhythm. While the participant reported being able to generate works which were typical of his normal approach, he also described himself as "relating" to the system rather than controlling it, saying "I didn't care whether I had control".

This participant valued that this high level, generative approach of the system prevented him from falling back on "previous ways of working" and made him do "things you wouldn't usually do". He highly valued the moments of "discovery" as the system generated variation over a number of bars, which was made possible by ceding control of detail: "I think they were coming alive.

UI, Basic Control and Navigation

All participants reported feeling more in control of the system in its final version. All indicated that the usability of this system was good, and much improved from the earlier prototype.

One factor in system clarity and ease of navigation was the improved descriptiveness of the UI. Participant 1 said "there was never a sense of needing to feel like you were confused by what any one node was doing" and attributed this in part to the addition of identifying colours for nodes and the audio "solo" function, which he used more as a diagnostic tool than a performance device.

All participants found and used all available options, and there were few questions about how aspects of the system worked. The exception here was the "Freedom" control for the quantiser system. This controlled overall system behaviour, but was situated found in a menu which controlled node-specific behaviours. All participants were confused by this and I had to explain the behaviour in each case.

Connection Weights

I found that users took one of two approaches to setting connection weights, and that one seemed more successful than the other. When users started by setting strong connections and adjusted from there they self-reported more success than when they began with quite low connection weights. In the latter case effect can be hard to discern, and the result of dragging connections quite subtle. Though no users directly commented on this, it might be counted as a usability issue for future attention.

Performance Control

Only one participant described approached using the system with a continuous performance in mind. This perspective revealed certain aspects of the system that others did not note.

Working in this way made the addition and removal of nodes a musical structuring device. One obstacle to this approach is Neurythmic introduces the nodes with a default pitch and volume. This feature was introduced to ease first contact with the system. However, participant 2 found that these sounds could be quite disruptive and dissonant when introduced to a system whose other voices had been changed, and with which the default sounds no-longer fitted. The result of this, he said was that "it feels like a risky move creating a node".

When asked about performance with the system, Participants 1 and 4 indicated that they would like a more immediate control method - both suggested multi-touch. Participant 1 was particularly enthusiastic about this, imagining moulding a network subtly with ten fingers on the screen. He also went further and speculated that a VR interface would be exciting for him.

Would participants use Neurythmic in their own work?

Two of the participants said they would use Neurythmic in their own music, and both independently asked if it might be possible to have a copy after the session. The third participant said he would most likely not use the system himself, and explained this as being because he only occasionally uses digital rhythm sequencing, and then in a quite idiomatic way. He did however have ideas about how other musicians of his acquaintance might use it, and his comments on the system were positive.

Potential Applications & Musical Context

All three participants in the final evaluation could see potential applications for the system, in their own work and in broader music culture.

Participant 1 described the system as "niche and practical" comparing it to a number of tools he and others used in music production and sound design work. He emphasised "niche" because it does a particular thing - in this case he felt it dealt with "chaos and order". He emphasised "practical" because he felt the system could be used in a wide range of contexts, both by experimental musicians and by people who will "use it as the starting point for very, very traditional music".

When asked about other systems inhabiting the same area he cited granular synthesis - a timbre synthesis and pattern generation approach developed by Curtis Roads [15]. He described Neurythmic as a "surface generator" in that same class. He also noted the growing market for experimental music tools like Neurythmic, citing Native Instruments' Woosh synthesizer and Spiral sequencer as systems in the same area.

Participants 2 and 1 both said they could imagine using the system for live performance. When testing the first prototype both suggested doing so with other musicians, participant 2 imagining this with "a second laptop slaved up to it", and participant 4 imagining playing with, for example, an improvising cellist he often works with.

In the second session Participant 2 commented that he would like to be able to record a multitrack output from the system to a DAW, so that he could generate material and later edit it.

Summary

Neurythmic takes an unusual approach to rhythm sequencing, we might expect it to have a slightly higher learning curve than conventional sequencers, but after two relatively brief testing sessions at least two of the three participants here considered it intuitive to work with. Both of these users requested a copy of the beta release of the software when it is complete.

Regarding novel qualities of the system: all users commented positively on the freedom of the system from quantisation, and even in the final version where a quantiser was available, most users used it only very lightly. Users quickly came to find the unusual spatial representation of rhythm-generation in the system intuitive. One user in particular saw value in this representation over the conventional, linear representation, which he said wasn't conducive to creating rhythms with "life and groove".

The sounds available in the system were intentionally restricted to focus evaluation on novel pattern generation features. This proved restrictive on some of the approaches participants tried to pursue and the improvement of options here is a priority for future development. Nonetheless even with the limited sound palette, participants demonstrated that the system was amenable to a wide array of approaches, some surprising to me.

Testing also revealed some avenues for future development, ranging from interface tweaks and bug fixes to a full VR rebuild - this is something I pick up in 13. In general, the issues users reported were with the particular implementation of Neurythmic itself, rather than the principles of rhythm generation with CPGs which it embodies. This indicates that this may be a fruitful area for future research.

Chapter 12
Network Patterns for Musical Applications

While working with the system and observing users, I found certain patterns of use gave good results. Neurythmic does not in the version presented here incorporate features to encourage use of these models. However, since there is little other literature on creative uses on CPGs I include details here as a descriptive resource, in the hope they may support future research in this area.

Sub-entrainment connections

Aside from [2], which was also concerned with musical applications, literature on applications of CPGs does not tend to discuss the use of sub-entrainment input weights. By contrast I found it useful to use connections both above the entrainment threshold and below.

Connections below entrainment are useful for musical applications due to their tendency to create patterns which may synchronise to the input signal, but repeating over several cycles of the input rather than one. Such patterns are described a little further in ch.7. This allows the creation of more complex patterns than is straightforward when working only above the entrainment threshold. In combination with some of the patterns below this is a fundamental mechanism of rhythmic generativity in the system described in this thesis.

One to Many

I found one to many, feedforward relationships useful in working with CPGs since they offer a means of coordination and grouping. This allows multiple rhythms to reference to a single "parent" pulse generated by one node. If that "parent" is itself generating a rhythm slightly more complex than a steady pulse, then results can quickly become complex and interesting, while retaining clear patterning.

Many to One

Many to one relationships are useful since they allow patterns to be generated where one voice's rhythm is not straightforwardly synchronised to a single other voice, but synchronises to two (or more), dependent on the coincidence and divergence of their cycles. Where one of a pair of signal inputs is very much slower than the other, this be used to generate an occasional variation on a beat which is the rest of the time synchronised to a faster beat. Though not explored in this project, this particular pattern might be extended by use of waveshaping on the signal connections, to reduce the effect of of a slow input during most of its cycle.

Many other variations on this pattern are possible, both simple - where results are quite easily to anticipate - and more complex - offering more possibilities for descovery and demanding a more reactive approach to control.

In Neurythmic the spatial representation of the network allows the simultaneous control of the weights of multiple connections into a single node. This allows a further development of the many-to-one pattern. It can easily be arranged so that moving a node increases one connection weight as it decreases another. This allows the voice's pattern to transition between two distinct behaviours, often with interesting results in the middle region where the two inputs interact. I found this a very useful performance technique and observed other users taking the same approach without prompting.

Sub-Networks

Sub-networks can be useful in a number of ways. More than one participant in the studies described in this thesis found that organisation by sub-networks arose naturally from the Neurythmic interface. This is because proximity influences connection strength and so lines of synchronisation, so nodes with like behaviour tend to group together on screen. This can be used as a tool for mental organisation, but also to create specific effects. I give two examples below.

First, bu using two or more sub-networks connected by a zero-weight connection, we can create unsynchronised sub-ensembles, and use these together to explore polyrhythmic or polytempo patterns. These patterns can then be partially or wholly synchronised by increasing connection weight between the sub-networks. This allows for fluid movement beween rhythmic complexity or pseudo-stochastic effects, and simple, easily read patterns.

Another approach is to use subnetwork organisation to create sub-ensembles in which feedback relationships result in near-inactivity. External signals into this sub-ensemble can then be used to "awaken" the group as a whole.

All these approaches allow for the specification of relatively complex generative patterns using smaller units which are simpler to understand and predict.

Burst generation

By using a slow signal input, into a fast node, at sub-entrainment strength, we can create bursts of rhythm from the faster node. If the incoming signal is high enough it will slow or stop the activity of the faster node through most of the incoming cycle, and and allow it to fire only during a brief period. With multiple inputs complex burst patterns can be generated. Again it might be interesting to apply waveshaping of signals (e.g. a power function) when using this pattern.

Prime or irrational beat bivisions

In a conventional sequencer, irrational divisions and multiples of the tempo, and rhythmic ratios whose lowest-common divisors are above ˜11 produce results in which periodicity is very hard to discern and which appear arrhythmic. In CPG based approaches this issue is overcome through entrainment and sub-entrainment effects which synchronise patterns. By use of these kinds of beat divisions and multiples, and control of connection weights it becomes possible to create expressive transitions from from arrhythmic-appearing spaces (at low connection weights) into tight, coherent rhythm (at weights close to, or at entrainment).

Chapter 13

Discussion and Future Directions

This thesis has described the development of Neurythmic, a tool for creating music with CPGs. This is the first software of its kind, and as such the thesis stands both as a description of a particular system, and of a model for further similar applications of CPGs in the area of musical creativity.

As well as delivering the Neurythmic application, I have developed a CPG library for MAX/MSP allowing non-programmer musicians to begin to explore the music making with CPGs, and provided some guidance on strategies in this area which I found successful.

I have also delivered two results which may be of use outside of creative applications of CPGs. In section 6 I described the method I implemented to accurately control the natural frequency of Matsuoka's Neural Oscillator(MNO) (building on [12]). I also present an equation for predicting the stopping threshold of MNO, alongside descriptions of the oscillator's behaviour on stopping and starting. These are results I was not able to find in the existing literature.

Future Work

My own intention in coming months is to port the system to iOS and implement a few of the features described below making Neurythmic available to a wider audience. Two of the participants involved in formal testing requested copies of Neurythmic, as have others who have used the system, so I am hopeful of at least a little interest.

Improvements to Neurythmic

Since the present version of Neurythmic focuses on particular testing scenarios, some features, such as limited sound generation, geared to evaluation, will be changed. There are also several other opportunities to develop Neurythmic further.

Multitouch and VR More than one study participant suggested that the system would benefit from a multitouch interface, allowing more immediate interaction. One user went a step further and suggested developing a VR interface, with 3d networks. This last idea might be combined with 3D projection techniques to create a performance system.

Connectivity Again, more than one study participant requested the ability to connect Neurythmic to other software or hardware, and to record the multitrack output of the system for later editing. Adding MIDI and/or OSC output would meet these requirements.

Synchronisation to DAW One study participant suggested that it would be interesting to perform with the system with another laptop musician, with the two systems

"slaved". It should be quite simple to implement this. In general terms, a phase ramp could be generated from an incoming clock signal, and that phase ramp used to read a table containing the waveform of MNO. Other nodes could then entrain to this signal.

Stopping and starting nodes This thesis describes the means to start and stop nodes accurately, but it was not possible within the time constraints of this project to make use of this. Several options present themselves - for example a touch interface might allow users to "pause" nodes by, e.g. holding them, allowing more immediate interaction with the system. Alternatively a different kind of signal connection could be created which sends an on/off signal to other nodes - for example every N cycles of the node. This would allow for quite complex variation and phrasing structures which are not possible with the system at the moment.

Robotics CPGs are most widely used in robotics, and [13] argues that hands-on tuning of these networks is sometimes the best approach. The approach taken to manual tuning in that paper is highly specific to a particular robot, and so a more lightweight, pluggable approach might be valuable. If so then it may be possible to develop such an approach building on Neurythmic - which in testing to date has shown itself to support intuitive interaction with these networks.

Other Musical Uses of CPGs

Beyond improvements to the Neurythmic application there are various other avenues for research into musical use of CPGs. Many of these could be explored using the MAX/MSP MNO library developed as part of this thesis.

Audio Synthesis One obvious research route is to explore audio synthesis with CPGs, perhaps using a similar interface and network structures to those described here. This picks up research begun in [2] but seemingly taken no further. This may work particularly well for parameter spaces in which MNO approximate a sine-wave. This seems likely to allow quite intuitive specification of timbral transitions moving between simple, harmonically related sonic spectra, oscillator-sync effects and harmonic noise.

Audiovisual Installation I began this project intending to build an audiovisual installation using CPGs, my original research review was written to this end, and for much of the initial period of prototyping I worked towards this goal. I moved away from this when I saw stronger research outcomes in the current form of the project. Nonetheless I still feel that CPG networks would lend themselves well to large scale multi-user interaction and the generation of spectacle. A room-sized, interactive, Central Pattern Generator, generating light and sound still holds great appeal.

Other Approaches In prototyping I briefly investigated the possibility of deriving phase ramps from oscillator outputs. While the approach was not suited to this case, applications using less complex networks may work well. Phase ramps so generated could be used to drive event sequencers, or other musical systems.

To detail just one option in this area, an approach closer to common uses in robotics might be imagined. A simple network of a handful of nodes, each driving and receiving input from force-feedback controllers, and each driving a note-event lookup. Something like this might form the start of a multitrack sequencer with tactile holistic control of expression on individual tracks.

Appendices

Appendix A
Task description for Explicitation study

This tasks explanation sheet is accompanied by the Information sheet. If you have not received the information sheet, please remind the experimenter to give you a copy.

Introduction

The purpose of this sheet is to explain the procedure of the experiment. My name is Daniel Bennett and I will be carrying out this experiment with you. Thanks in advance for your help.

Please read just the first side of this sheet for now.

Part 1 Creative Musical Task on a Familiar Instrument

In the first part of the experiment you will be asked to perform the creative, musical task described below. You can do this on a musical instrument familiar to you, of your choosing. I will film and audio-record this process.

Afterwards you will be encouraged to talk about the task - Id like you to focus on and describe details of the experience, and Ill occasionally interrupt with questions focusing on specific moments. After reading this side of the sheet, and before you start the task I will discuss this stage a little with you, and make sure youre comfortable with everything. If you have any questions, please ask.

The Task

Please imagine you have been asked by a writer to generate a brief sonic mood board as a preliminary stage in developing the audio for their television show. The writer does not want to see or hear footage of this session specifically just use this session to work towards creating that material.

The writer has chosen you for this task because they know your work well. The television show they have asked you to work on is well suited to your musical style, and you may imagine the show in any way you like. The writer would like you to create material representative of your normal approach, but is keen to hear new material, something first developed in this session.

We have put aside 15-30 minutes for you to begin to develop this new material please feel free to take as little or as much of that as you like. Just work towards something you think is interesting.

Appendix B

Task description for Think Aloud Study

Part 2 Testing a Prototype Computer Music Instrument

In the second part of the experiment you will be asked to use and discuss a prototype computer music instrument.

This instrument is loosely like a drum machine or sequencer. It generates musical patterns based on the behavior of an artificial neural network, modeled on the biological neural networks that generate heartbeats, walking movements, breathing patterns.

In this instrument, each node in the network creates light and sound when it fires and the pattern of firing of each node is based on its relationship with other nodes, as illustrated by the interface.

You can interact with this instrument by

- changing the strength of the connections between the nodes
- adding and removing nodes and changing the connections of the network
- changing the natural firing rate of the nodes
- changing the sounds individual nodes generate

Appendix C
Task description for Final Evaluation

Introduction
The purpose of this sheet is to explain the procedure of the experiment.

Thanks in advance for your help.

Last time, in the first part of the session, you performed a creative, musical task, which you were asked to imagine was prompted by a television writer who knew your work well. Afterwards you were later interviewed about the task. This session will follow the same pattern as that first part.

As last time, first I will ask you to perform a musical task (described below), and afterwards I would like to interview you about the task.

In the interview, as last time, you will be encouraged to talk about your experience of the task. Ill encourage you to focus on, and describe particular details of the experience. Occasionally Ill interrupt with questions focusing on specific moments.

Once youre done reading this sheet, and before you start the task, well discuss the process to make sure youre comfortable with everything. Please ask any questions that come to mind.

The Task
Please imagine that the writer who commissioned the work in our last session has begun another project. This time a theme of the project is computational creativity, and so they would like you to make at least some of the music for the project using the Neurythmic system.

Again, the writer has come to you because they know your work well. They are interested in your particular creative approach to this system. Please imagine the show in any way you like whether as documentary or fiction, as serious or light entertainment, experimental, or conventional, or anywhere between or outside of that that would suit your musical approach.

We have put aside 15-30 minutes for you to begin to develop this new material and ideas ahead of this meeting. Please feel free to take as little or as much of that as you like. Focus on working towards something you think is interesting you need not come away with a complete piece, or imagine that recordings of the session will be presented at the meeting, unless that seems to you to be the best approach.

Appendix D
Code used for binary search to find Entrainment threshold

(listing D.1)

Listing D.1: search for entrainment threshold

```
double Tests_AtoB::t2_getEntrainmentThreshold(double t1Overt2, double
    bAndG,double freqRatio, double phase)
{
    // binary search for entrainment
    bool entrained = false;
    bool successThis = false, successPrev = false, refining = false;
    int attempts = 0, refinements = 0;
    double minWeight = T2_MIN_WEIGHT;
    double maxWeight = T2_MAX_WEIGHT;
    double weight = T2_GUESS_WEIGHT;
    while (!entrained && attempts < T2_ATTEMPTS_COUNT
            && refinements < T2_REFINEMENTS_COUNT
            && weight < T2_TOP_GIVEUP_WEIGHT && weight >
                T2_BOTTOM_GIVEUP_WEIGHT) {
        if (refining) {
            refinements++;
        } else {
            attempts++;
        }
        successThis = t2_checkIfEntrains(t1Overt2, bAndG, freqRatio, weight,
            phase);
        if (attempts > 1 && successThis != successPrev) {
            refining = true;
        }
        if (successThis) {
            maxWeight = weight;
        } else {
            minWeight = weight;
        }
        weight = ((maxWeight - minWeight) / 2) + minWeight;
    }

    if (refining) {return weight;}
    return 0.0;
}
```

Bibliography

[1] N. Collins and A. McLean, "Algorave: Live performance of algorithmic electronic dance music," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014, pp. 355–358 (cit. on p. 3).

[2] A. Eldridge. (2005). Neural oscillator synthesis: Generating adaptive signals with a continuous-time neural model, [Online]. Available: http://www.ecila.org/ecila_files/content/papers/ACEICMC05.pdf (visited on 04/14/2016) (cit. on pp. 3, 9, 73, 76).

[3] C. Roads, *The computer music tutorial*. 1996 (cit. on pp. 4, 56).

[4] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008 (cit. on p. 7).

[5] P. S. Katz and R. M. Harris-Warrick, "Neuromodulation of the crab pyloric central pattern generator by serotonergic/cholinergic proprioceptive afferents," *Journal of Neuroscience*, vol. 10, no. 5, pp. 1495–1512, 1990 (cit. on p. 7).

[6] M. M. Williamson, "Oscillators and crank turning: Exploiting natural dynamics with a humanoid robot arm," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1811, pp. 2207–2223, 2003 (cit. on p. 7).

[7] F. Fang, W. L. Xu, K. Lin, F. Alam, and J. Potgieter, "Matsuoka neuronal oscillator for traffic signal control using agent-based simulation," *Procedia Computer Science*, vol. 19, pp. 389–395, 2013 (cit. on p. 7).

[8] K. Matsuoka, "Sustained oscillations generated by mutually inhibiting neurons with adaptation," *Biological cybernetics*, vol. 52, no. 6, pp. 367–376, 1985 (cit. on pp. 7, 8, 10).

[9] ——, "Mechanisms of frequency and pattern control in the neural rhythm generators," *Biological cybernetics*, vol. 56, no. 5-6, pp. 345–353, 1987 (cit. on p. 8).

[10] H. Kerlleñevich, P. E. Riera, and M. C. Eguia, "Santiago-a real-time biological neural network environment for generative music creation," in *Applications of Evolutionary Computation*, 2011, pp. 344–353 (cit. on p. 9).

[11] D. Bisig and P. Kocher, "Early investigations into musical applications of time-delayed recurrent networks," in *Proceedings of the Generative Art Conference, Milano*, 2013 (cit. on p. 9).

[12] K. Matsuoka, "Analysis of a neural oscillator," *Biological cybernetics*, vol. 104, no. 4-5, pp. 297–304, 2011 (cit. on pp. 10, 11, 21, 22, 75).

[13] F. Dalla Libera, T. Minato, H. Ishiguro, and E. Menegatti, "Direct programming of a central pattern generator for periodic motions by touching," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 847–854, 2010 (cit. on pp. 11, 76).

[14] C. Roads, *Composing electronic music: A new aesthetic*. 2015 (cit. on pp. 12–14).

[15] ——, *Microsound*. 2004 (cit. on pp. 12, 72).

[16] (2017). Elektron - oktatrack performance sequencer, [Online]. Available: https://www.elektron.se/products/octatrack/ (visited on 08/25/2017) (cit. on p. 13).

[17] (2017). Makenoise - rene sequencer module, [Online]. Available: http://www.makenoisemusic.com/modules/rene (visited on 08/25/2017) (cit. on p. 14).

[18] G. T. Toussaint *et al.*, "The euclidean algorithm generates traditional musical rhythms," in *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, 2005, pp. 47–56 (cit. on p. 14).

[19] A. McLean, "Making programming languages to dance to: Live coding with tidal," in *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modelling and Design*, ACM, 2014 (cit. on pp. 14, 15).

[20] J. Harriman, "Sinkapater-an untethered beat sequencer.," in *New Interfaces for Musical Expression*, 2012 (cit. on p. 15).

[21] U. Brandstatter, M. Brandstatter, and C. Sommerer, "Gears for audio-visual composition: Productive play with virtual mechanics," in *Proceedings of the Audio Mostly 2016*, ACM, 2016, pp. 170–177 (cit. on p. 15).

[22] S. Bodker, "Scenarios in user centred design setting the stage for reflection and action," *Interacting with computers*, vol. 13, no. 1, pp. 61–75, 2000 (cit. on p. 16).

[23] K. Andersen and D. Gibson, "The instrument as the source of new in new music," *Design Issues*, 2017 (cit. on pp. 16, 17, 41).

[24] N. Schnell and M. Battier, "Introducing composed instruments, technical and musicological implications," in *Proceedings of the 2002 conference on New interfaces for musical expression*, National University of Singapore, 2002, pp. 1–5 (cit. on p. 16).

[25] A. Johnston, "Opportunities for practice-based research in musical instrument design," *Leonardo*, vol. 49, no. 1, pp. 82–83, 2016 (cit. on p. 16).

[26] R. Rasch, "Tuning and temperament," *The Cambridge history of Western music theory*, pp. 193–222, 2002 (cit. on p. 16).

[27] J. Pritchett, "From choice to chance: John cage's concerto for prepared piano," *Perspectives of New Music*, pp. 50–81, 1988 (cit. on p. 16).

[28] O. W. Bertelsen, M. Breinbjerg, and S. Pold, "Instrumentness for creativity mediation, materiality & metonymy," in *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, ACM, 2007, pp. 233–242 (cit. on pp. 17, 45, 47, 56, 58).

[29] ——, "Tool–material, metaphor–metonymy, instrument (ness)," in *Proceedings of the Eighth Danish Human-Computer Interaction Research Symposium*, 2008 (cit. on p. 17).

[30] ——, "Instrumentness in clusters of artefacts–a first take on collaborative creativity," in *Proceedings of the 6th Conference on Creativity and Cognition*, 2007 (cit. on p. 17).

[31] G. Moody. (2014). WFDB applications guide, [Online]. Available: https://www.physionet.org/physiotools/wfdb/psd/ (visited on 04/28/2016) (cit. on p. 20).

[32] O. Niemitalo. (2008). Polynomial interpolators for high-quality resampling of oversampled audio, [Online]. Available: http://yehar.com/blog/wp-content/uploads/2009/08/deip.pdf (visited on 04/28/2016) (cit. on p. 21).

[33] A. M. Al-Busaidi, R. Zaier, and A. S. Al-Yahmadi, "Control of biped robot joint angles using coordinated matsuoka oscillators," in *International Conference on Artificial Neural Networks*, Springer, 2012, pp. 304–312 (cit. on pp. 23–26).

[34] P. Nash. (2017). Catch: Behavioural driven development in c++, [Online]. Available: https://github.com/philsquared/Catch (visited on 08/25/2017) (cit. on p. 41).

[35] A. M. Signatories. (2001). The agile manifesto, [Online]. Available: http://agilemanifesto.org/ (visited on 08/25/2017) (cit. on p. 41).

[36] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006 (cit. on pp. 42, 68).

[37] J. Nielsen. (2012). Thinking aloud: The no 1 usability tool, [Online]. Available: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/ (visited on 08/28/2017) (cit. on p. 43).

[38] A. M. Gill and B. Nonnecke, "Think aloud: Effects and validity," in *Proceedings of the 30th ACM international conference on Design of communication*, ACM, 2012, pp. 31–36 (cit. on p. 43).

[39] M. Obrist, S. A. Seah, and S. Subramanian, "Talking about tactile experiences," in *Proceedings. SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2013, pp. 1659–1668 (cit. on p. 44).

[40] M. Maurel, "The explicitation interview: Examples and applications," *Journal of Consciousness Studies*, vol. 16, no. 10-1, pp. 58–89, 2009 (cit. on pp. 44–46).

[41] J. Nielsen. (2012). Demonstrate thinking aloud by showing users a video, [Online]. Available: https://www.nngroup.com/articles/thinking-aloud-demo-video/ (visited on 08/28/2017) (cit. on p. 45).

[42] P. Vermersch, "Describing the practice of introspection," *Journal of Consciousness Studies*, vol. 16, no. 10-1, pp. 20–57, 2009 (cit. on p. 46).

[43] A. Harley. (2014). Icon usability, [Online]. Available: https://www.nngroup.com/articles/icon-usability/ (visited on 08/25/2017) (cit. on pp. 50, 59).

[44] R. Boulanger, V. Lazzarini, and M. Mathews, *The audio programming book*. 2010 (cit. on p. 56).

[45] D. Norman, *The design of everyday things: Revised and expanded edition*. 2013 (cit. on pp. 57, 58).

[46] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, "Automatic music transcription: Challenges and future directions," *Journal of Intelligent Information Systems*, vol. 41, no. 3, pp. 407–434, 2013 (cit. on p. 62).

[47] B. H. Repp, "Detecting deviations from metronomic timing in music: Effects of perceptual structure on the mental timekeeper," *Attention, Perception, & Psychophysics*, vol. 61, no. 3, pp. 529–548, 1999 (cit. on p. 65).